File Number 7090-27
Form C28-6275-2

IBM Systems Reference Library

IBM 7090/7094 IBSYS Operating System
IBJOB Processor

VERSION 11

This publication describes the 7090/7094 IBJOB Processor and its components. Included are descriptions of the IBJOB Processor, the functions that are performed by the components of the Processor, and the requirements for use of the Processor and its components.

The IBJOB Processor, 7090-PR-929, is a group of programs used to translate programming languages. It consists of the following:

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

This publication is divided into two parts. The first part contains information for the applications programmer, and the second part contains information for the systems programmer.

Preface

This publication provides procedural information required to compile, assemble, load, and execute a program, using the IBJOB Processor. Also included is information intended for the experienced 7090/7094 programmer and the systems programmer.

It is assumed that the reader is familiar with the contents of the publication *IBM 7090/7094 IBSYS Operating System: System Monitor (IBSYS)*, Form C28-6248.

The reader should also be familiar with one or more of the publications that describe the languages associated with the Assembler and the compilers that operate within the IBJOB Processor. These publications are:

IBM 7090/7094 Programming Systems: FORTRAN IV Language, Form C28-6274.

IBM 7090/7094 Programming Systems: COBOL Language: Preliminary Specifications, Form J28-6260.

IBM 7090/7094 Programming Systems: Macro Assembly Program (MAP) Language, Form C28-6311.

The MAP programmer who uses the Input/Output Control System should also read the publication *IBM 7090/7094 IBSYS Operating System: Input/Output Control System*, Form C28-6345.

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. If an IBM 1401 with its attached 1402 Card Read Punch and 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), then only seven units are required.

Required for installations using IBM 1301 Disk Storage or IBM 7320 Drum Storage:

Four tape units. 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 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 three units are required.

Five units. IBM 729 Magnetic Tape Units, IBM 7340 Hypertape Drives, or selected cylinders of IBM 1301 Disk Storage or of IBM 7320 Drum Storage can be assigned to the installation in any combination.

MAJOR REVISION (February, 1964)
This publication, Form C28-6275-2, supersedes the publication *IBM 7090/7094 Programming Systems: IBJOB Processor*, Form C28-6275-1; the supplements *IBM 7090/7094 Programming Systems: IBJOB Processor: Overlay Feature of IBLDR*, Form C28-6331 and *IBM 7090/7094 Programming Systems: IBJOB Processor—7090-PR-929: Supplement for 1301 Disk Capability*, Form C28-6325; and the Technical Newsletters Form N28-0056 and Form N28-0060.

Copies of this and other IBM publications can be obtained through IBM Branch Offices.
Address comments concerning the contents of this publication to:
IBM Corporation, Programming Systems Publications, Dept. D91, PO Box 390, Poughkeepsie, N.Y. 12602

Contents

Introduction

Organization of This Publication

This publication is divided into two parts. The first part contains information for the applications programmer, and the second part contains information for the systems programmer.

A programmer who wishes to make full use of all the features of the IBJOB Processor would normally read the section "Programmer's Information" from beginning to end.

However, the section "Programmer's Information" is organized so that the MAP, FORTRAN, or COBOL programmer, who requires only a basic knowledge of the system to run a simple job, need only read the section "The Processor Monitor" (omitting information pertaining to the Input/Output Editor) and the sections dealing with the control cards for the specific language. When a programmer has familiarized himself with the control card formats, he need only refer to the control card checklist and the control card format index in the appendixes to this publication.

General Description of the IBJOB Processor

The IBM 7090/7094 Operating System provides monitored control for the systems programs written for the 7090 and the 7094. The overall control is provided by the System Monitor (IBSYS), which contains the routines and data necessary for continuous system operation. In addition, the Processor Monitor provides a second level of monitoring for the IBJOB Processor, which consists of a group of programs used to translate programming languages and to permit the loading and execution of the compiled and assembled programs. The IBJOB Processor consists of the following:

The Processor Monitor (IBJOB)
The FORTRAN IV Compiler (IBFTC)
The COBOL Compiler (IBCBC)
The Assembler — The Macro Assembly Program (IBMAP)
A relocating loader — The Loader (IBLDR)
Preassembled subroutines to be used, if required, by the object program — The Subroutine Library (IBLIB)

These programs are interdependent, since the output from each compiler is processed by the Macro Assembly Program (the Assembler), and the output from the Assembler is processed by the Loader.

The Processor Monitor reads control cards that specify the action to be performed in a Processor application. A Processor application is the basic unit of work that can be performed by the IBJOB Processor. An application can consist of one or more compilations, assemblies, or the loading of relocatable programs that were assembled previously. In this way, a single program can be written, requiring the use of several IBJOB Processor components. The interdependency of the IBJOB Processor components is shown in Figure 1.

Input/Output Control System (IOCS)

The input/output routines of IOCS, which may be used by an object program prepared by the IBJOB Processor, are located in the Subroutine Library. The input/output routines are divided into groups of routines, i.e., each input/output routine is not a separate entity in the Subroutine Library. These groups of routines are defined as levels of IOCS. The levels of IOCS contained in the Subroutine Library are: the Minimum level, the Basic level, and the Labels level. In addition to the levels of IOCS, Random IOCS is also located in the Subroutine Library. The routines contained in each level of IOCS and in Random IOCS are described in the publication *IBM 7090/7094 IBSYS Operating System: Input/Output Control System,* Form C28-6345.

The programmer need not specify the level of IOCS required by his object program. The calling sequences that are written by the MAP programmer or generated by the compilers and the specifications in Loader control cards supply sufficient information to the Loader to load the required level with the object program. If more IOCS facilities are desired, the desired level may be specified in a control card (further information may be found in the section "$IBJOB Card").

The Minimum level of IOCS has been modified for use by FORTRAN IV object programs and placed in the Subroutine Library. The routines for the Minimum level and for the modified package (called FORTRAN IOCS) are the same, but many of the error-checking features and messages present in the Minimum package have been deleted from FORTRAN IOCS, since they are not required for FORTRAN IV object programs.

Core Storage Allocation

Figure 2 shows the core storage arrangement of the major components of the Operating System during a Processor application. Also shown is the core storage location of the object program during execution.

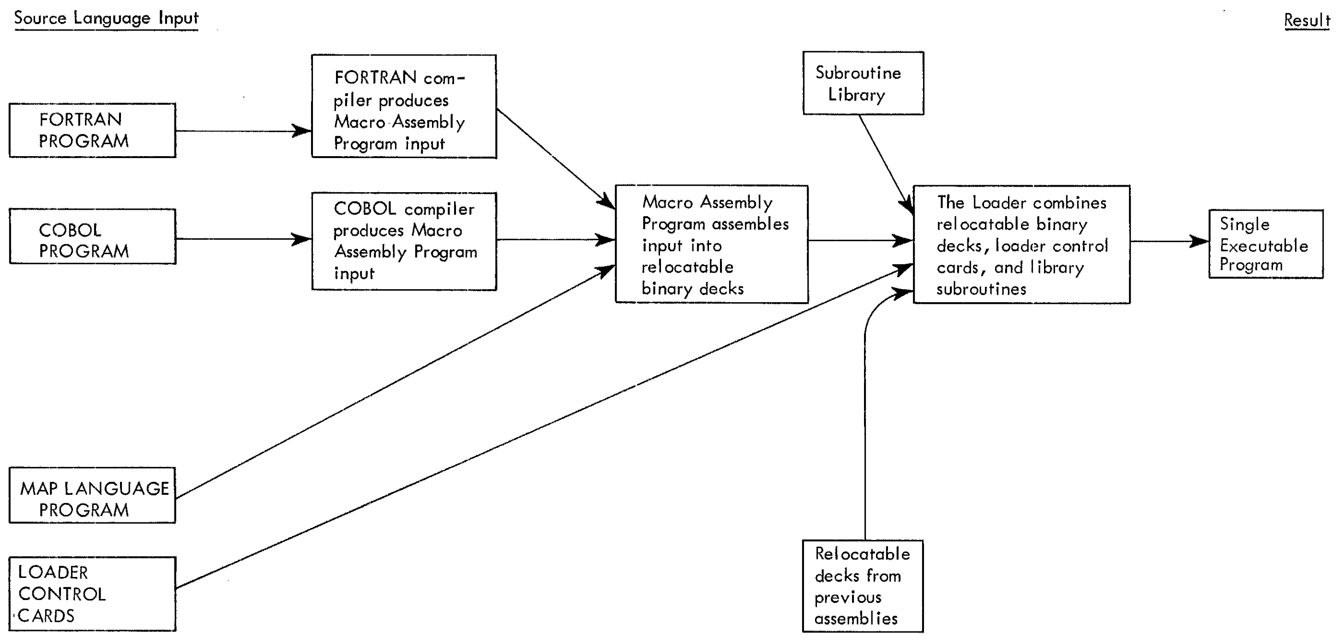

Source Language Input
Result
FORTRAN PROGRAM
COBOL PROGRAM
MAP LANGUAGE PROGRAM
LOADER CONTROL CARDS
FORTRAN compiler produces Macro Assembly Program input
COBOL compiler produces Macro Assembly Program input
Macro Assembly Program assembles input into relocatable binary decks
Subroutine Library
The Loader combines relocatable binary decks, loader control cards, and library subroutines
Relocatable decks from previous assemblies
Single Executable Program

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

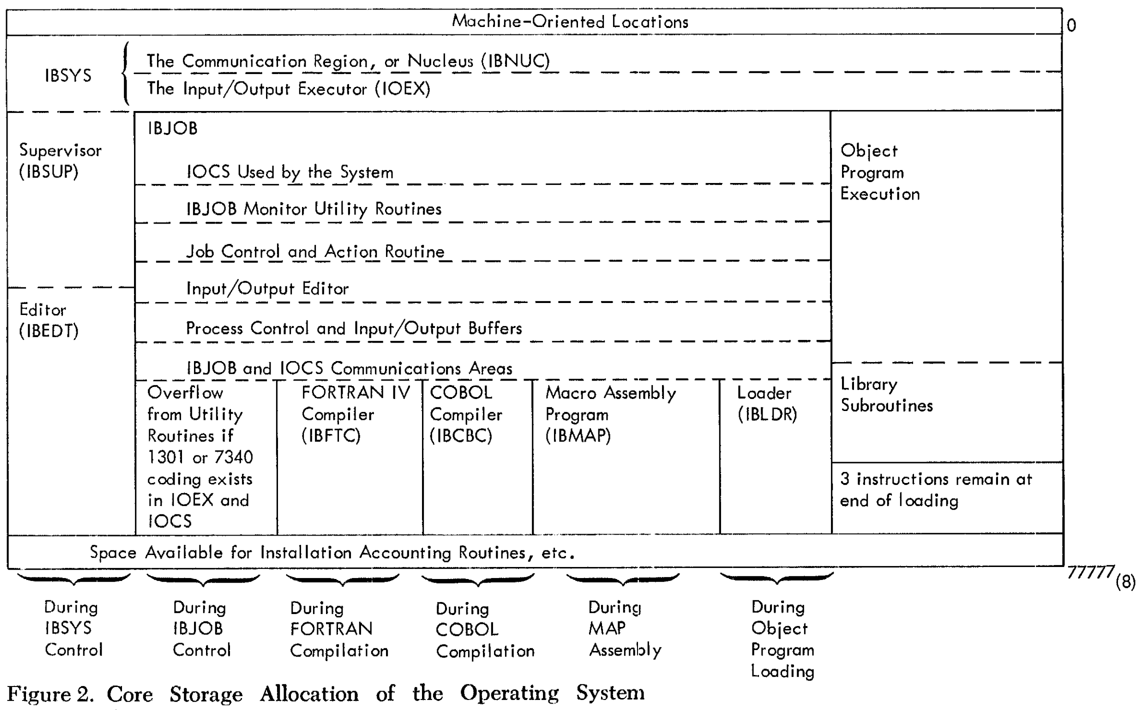

Machine-Oriented Locations
0
IBSYS
The Communication Region, or Nucleus (IBNUC)
The Input/Output Executor (IOEX)
Supervisor (IBSUP)
IBJOB
IOCS Used by the System
IBJOB Monitor Utility Routines
Job Control and Action Routine
Editor (IBEDT)
Input/Output Editor
Process Control and Input/Output Buffers
IBJOB and IOCS Communications Areas
Overflow from Utility Routines if 1301 or 7340 coding exists in IOEX and IOCS
FORTRAN IV Compiler (IBFTC)
COBOL Compiler (IBCBC)
Macro Assembly Program (IBMAP)
Loader (IBLDR)
Object Program Execution
Library Subroutines
3 instructions remain at end of loading
Space Available for Installation Accounting Routines, etc.
77777(8)
During IBSYS Control
During IBJOB Control
During FORTRAN Compilation
During COBOL Compilation
During MAP Assembly
During Object Program Loading

Figure 2. Core Storage Allocation of the Operating System Components

The Processor Monitor

The Processor Monitor is the supervisory portion of the IBJOB Processor. It provides communication between the System Monitor and the components of the IBJOB Processor. The Processor Monitor is called into core storage when the System Monitor reads a $EXECUTE card with IBJOB in the variable field. The Processor Monitor then reads the next control card.

Control Card Notation

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

1. Material in brackets [ ] represents an option that may be omitted or included, at the programmer's choice.
2. Material in braces { } indicates that a choice of the contents is to be made by the programmer. The standard option, which is underlined, is assumed when an option is not specified.
3. Upper-case specifications must be present in the form specified, if used.
4. Lower-case specifications represent typical quantities or terms whose value(s) must be supplied by the programmer.
5. The order in which options are specified on the various control cards is not critical, unless otherwise specified.
6. Commas are used to separate options, when options are present. If no options are present, a string of commas is not necessary to indicate the absence of options, unless otherwise specified.

Processor Monitor Cards Required for an Application

The following control cards are required for a Processor application.

$JOB Card

The Processor Monitor transfers to the System Monitor when it reads this control card. If units have not been reassigned or made unavailable during the last job and a between-jobs interrupt condition does not exist, the Processor Monitor regains control and transfers to the installation accounting routine. If there is no installation accounting routine, the $JOB card is printed online. If units have been reassigned or made unavailable during the last job or if a between-jobs interrupt condition exists, the System Monitor processes the card and retains control until a card is read that calls for a subsystem to be loaded.

$EXECUTE Card

This card must precede a Processor application within a job, if one of the following conditions is present:

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 system name in column 16. If the name is IBJOB, no action is taken. If the name is anything other than IBJOB, this information is given to the System Monitor. The Processor Monitor then transfers to the System Monitor.

$IBJOB Card

The $IBJOB card must be the first control card read by the Processor Monitor for a given application. The options that can be specified in this control card describe the manner in which an application is to be processed. The format of the $IBJOB card is:

1 16

$IBJOB

$$\left[\begin{Bmatrix}\underline{\text{GO}}\\ \text{NOGO}\end{Bmatrix}\right] \left[,\begin{Bmatrix}\underline{\text{NOLOGIC}}\\ \text{LOGIC}\\ \text{DLOGIC}\end{Bmatrix}\right] \left[,\begin{Bmatrix}\underline{\text{NOMAP}}\\ \text{MAP}\end{Bmatrix}\right]$$
$$\left[,\begin{Bmatrix}\underline{\text{NOFILES}}\\ \text{FILES}\end{Bmatrix}\right] \left[,\begin{Bmatrix}\underline{\text{SOURCE}}\\ \text{NOSOURCE}\end{Bmatrix}\right]$$
$$\left[,\begin{Bmatrix}\underline{\text{IOEX}}\\ \text{MINIMUM}\\ \text{BASIC}\\ \text{LABELS}\\ \text{FIOCS}\end{Bmatrix}\right] \left[,\begin{Bmatrix}\underline{\text{FLOW}}\\ \text{NOFLOW}\end{Bmatrix}\right]$$

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

EXECUTION OPTIONS

The execution options are:

1. GO — The object program is to be executed after it is loaded.
2. NOGO — The object program is not executed, even if it is loaded.

If NOGO 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.

LOGIC OPTIONS

The logic options are:

1. LOGIC — A cross-reference table of the program sections and 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.

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

3. NOLOGIC — A cross-reference table is not wanted.

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

MAP OPTIONS

The MAP options are:

1. MAP — A core storage map, giving the origin and the amount of 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.

2. NOMAP — A core storage map is not wanted.

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

FILE LIST OPTIONS

The file list options are:

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

2. NOFILES — The list is printed on-line, but is not written off-line.

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

INPUT DECK OPTIONS

The input deck options are:

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

2. 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 or assembly is in the application.

IOCS OPTIONS

The IOCS options are:

1. IOEX — The object program uses the Input/Output Executor (IOEX) only.

2. MINIMUM — The minimum-level package of IOCS is to be loaded with the object program.

3. BASIC — The basic-level package of IOCS is to be loaded with the object program.

4. LABELS — The labels-level package of IOCS is to be loaded with the object program.

5. FIOCS — The IOCS package intended especially for use by FORTRAN IV object programs is loaded with the object program.

If none of these options is specified, it is assumed that the object program uses the Input/Output Executor.

These specifications may be ignored by the Loader at load time, if the object program requires a higher level of IOCS than is specified.

OVERLAY OPTIONS

The overlay options are:

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

2. 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.

Component Control Cards

Each component operating within the IBJOB Processor has a unique control card that causes the Processor Monitor to call for the loading of the component. In this way, the Processor Monitor controls the transition from one type of Processor application to another (e.g., a FORTRAN compilation followed by a COBOL compilation).

End-of-File Card

This card must be the last card in a Processor application. Either a card with a 7 and 8 punch in column 1 or any control card that causes file mark to be written by a peripheral program can be used as an end-of-file card.

Terminating Object Program Execution

Execution of an object program can be terminated by a transfer (TRA) to .LXRTN or to IBEXIT, by a call to the entry point EXIT, or by using the SAVE and the RETURN pseudo-operations. When execution is terminated, control is returned to the System Monitor.

Optional Processor Monitor Cards

$IBSYS Card

The Processor Monitor prints the message "RETURNING TO IBSYS" on-line when the $IBSYS card is read, and transfers to the System Monitor.

$ID Card

The Processor Monitor transfers to the installation accounting routine. If this control card is read and there is no installation accounting routine, the Processor Monitor prints the $ID card on-line.

$STOP Card

The Processor Monitor indicates to the System Monitor that a $STOP card has been read and transfers to the System Monitor, which processes this control card.

$PAUSE Card

This control card causes the machine to stop, and the contents of the card are printed on-line. The variable field of the control card should contain a message that explains to the operator why the machine has stopped. Another control card must follow this control card.

$* Card

This control card is used as a comments card. It causes no action, other than being printed on-line and listed off-line.

$ENTRY Card

This control card specifies the location of the initial transfer to the object program at execution time. The variable field contains a literal, consisting of 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 to either the standard entry point of the first deck retained or to an entry point whose name is ......' (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 exist:

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

This control card indicates the beginning of a data file on the input unit. This card can be replaced by an end-of-file card. The data file must also be followed by an end-of-file card.

$ENDREEL Card

This control card causes a reel switch involving the System Input Unit (SYSIN1) and the secondary System Input Unit (SYSIN2). It must be preceded by an end-of-file card. This control card cannot be placed in the middle of a data file.

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 on the $IEDIT card. 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 by using the $OEDIT card.

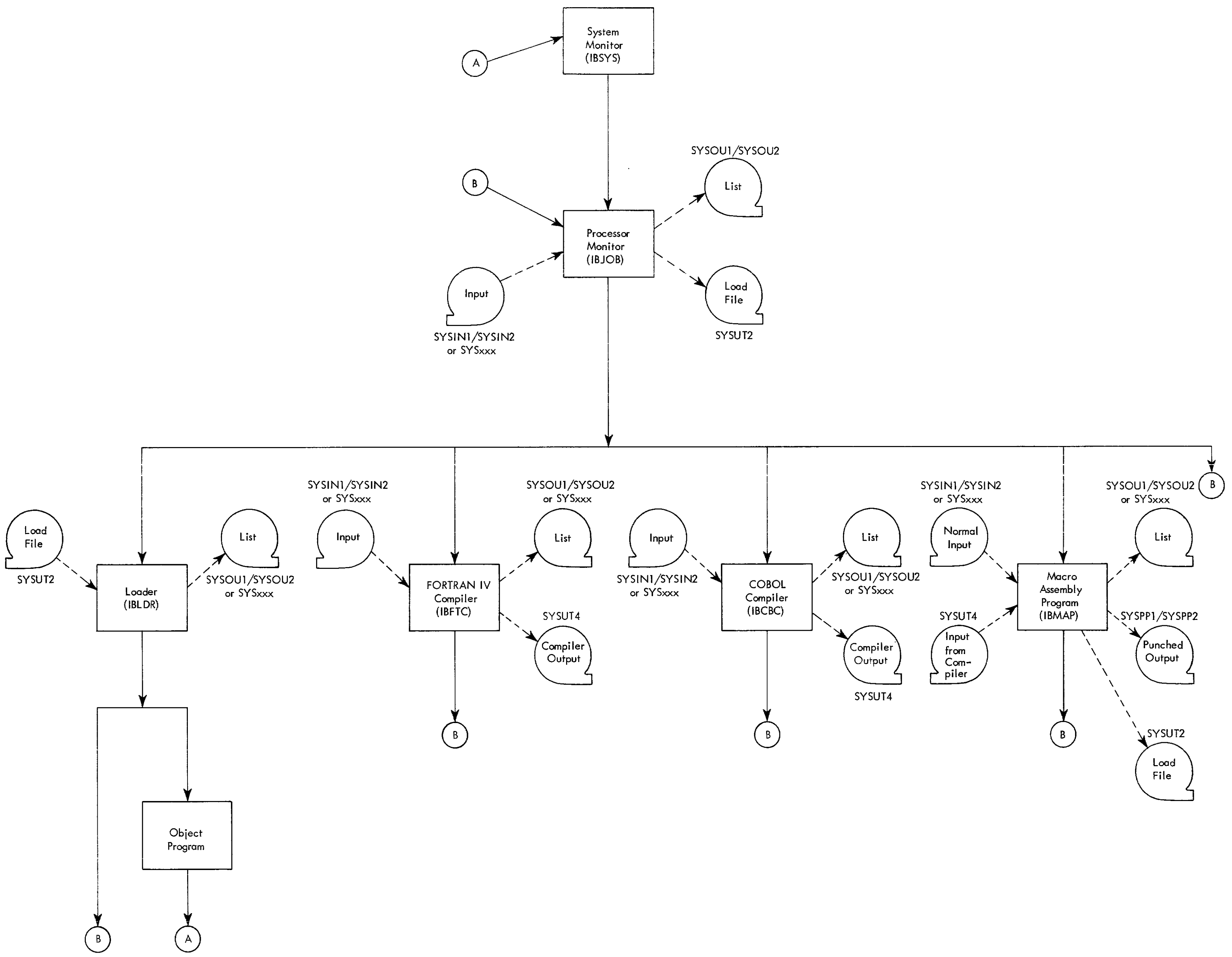
The Input/Output Editor also writes the output from both compilers and reads the input for the Assembler and the Loader.

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



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

$IEDIT and $OEDIT cards and explanations of their options are given in the following text.

$IEDIT Card

The contents of this control card set input specifications either for the remainder of the application or until another $IEDIT card is read. The format of the $IEDIT card is:

```
1             16
$IEDIT        [{SYSIN1}]  [ {NOSRCH} ]  [, {NOALTER}]
              [{SYSxxx}]  [,{SRCHn } ]  [  {ALTER  }]
                          [ {SCHFn } ]
```

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

INPUT OPTIONS

The input options are:

1. SYSIN1 — The source, symbolic, or object program immediately follows the component control card on the System Input Unit (SYSIN1).
2. 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.

SEARCH OPTIONS

The search options are:

1. NOSRCH — The specified alternate input unit is positioned correctly.
2. 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.
3. 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.

ALTER OPTIONS

The alter options are:

1. NOALTER — There are no Alter cards.
2. ALTER — There are Alter cards on the System Input Unit.

If neither NOALTER nor ALTER is specified, it is assumed that there are no Alter cards.

$OEDIT Card

The contents of this control card set output specifications either for the remainder of the application or until another $OEDIT card is read. The format of the $OEDIT card is:

```
1             16
$OEDIT        [{SYSOU1}]  [, {NOPREST}]  [, {NOCPR }]
              [{SYSxxx}]  [  {PREST  }]  [  {CPREST}]
```

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

OUTPUT OPTIONS

The output options are:

1. SYSOU1 — The output listings for this deck are placed in the System Output Unit.
2. 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, and SYSLB4).

If neither SYSOU1 nor an alternate output unit is specified, the output is written on the System Output Unit.

ASSEMBLER PREST OPTIONS

The Assembler Prest options are as follows:

1. PREST — A compressed form of the symbolic input to the Assembler is written on the System Peripheral Punch Unit. The compressed deck is called a Prest deck.
2. NOPREST — A Prest symbolic deck is not wanted.

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

COMPILER PREST OPTIONS

The compiler Prest options are:

1. CPREST — A Prest deck of the source input to either compiler is written on the System Peripheral Punch Unit. A $FTEND card must follow the END statement in a FORTRAN IV deck if a Prest source deck is desired. This control card is included in the Prest source deck.
2. NOCPR — A Prest source deck is not wanted.

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

If both PREST and CPREST are specified in the $OEDIT card that precedes a source deck, both compiler input and output are written on the System Peripheral Punch Unit in Prest form.

Altering an Input Deck

Any symbolic, source, or Prest input deck can be modified, and a new Prest symbolic deck or Prest source deck, which includes the changes, can be produced. 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 control cards must be on the System Input Unit. The input deck must be preceded by a component control card. The Alter control cards on the System Input Unit must also be preceded by a component control card of the same type and with the same deck name.

Use of 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 (label field) of a card are normally printed. The numbers are right-justified, sequential digits with leading blanks, and consist of a maximum of eight digits.

If a source deck or a symbolic deck is to be altered, the existing label 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    n
```

Fields m and n are the contents of the label 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 n are the generated Alter number. The first blank character appearing in the label field indicates that all prior characters constitute field m. The characters remaining after the blank or blanks constitute field n. In the label field, field m is left-justified and field n is right-justified. If the label field contains no blank characters, then field m may be omitted or may consist of no more than the first six characters of the label field. Field n then consists of the remaining characters that were not placed in field m. Fields m and n must have a total number of characters equal to the number of characters in the label, excluding leading or embedded blanks. For example, if the label in columns 73-80 is LABEL090 then the format for this label, 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 label in columns 73-80 is LABELbb9, the format for this label is as follows:

```
1             8         16
LABEL         *ALTER    9
```

If there are embedded blanks in the label, 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 from 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

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

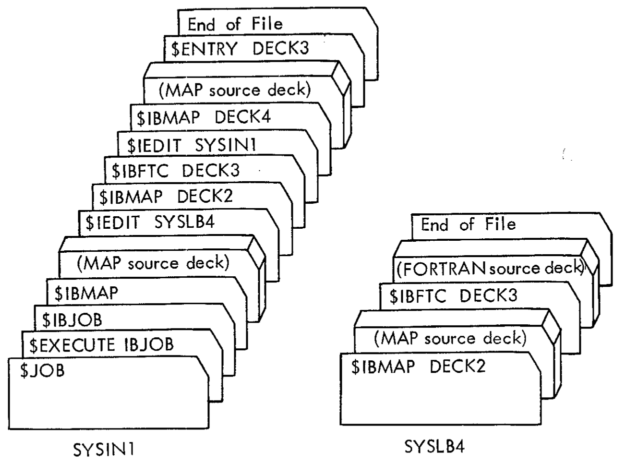


Figure 4. Sample Control Card Deck for Use of an Alternate Input Unit

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 control cards, to be read from the System Library Unit (SYSLB4).

3. The $IBMAP card, specifying DECK2, is read from the System Input Unit (SYSIN1). The corresponding $IBMAP card, specifying DECK2, is read from the System Library Unit (SYSLB4), and the MAP language deck on the System Library Unit (SYSLB4) is assembled.

4. The $IBFTC card, specifying DECK3, is read from the System Input Unit (SYSIN1). The corresponding $IBFTC card, specifying DECK3, is read from the System Library Unit (SYSLB4), and the FORTRAN IV language deck on the System Library Unit (SYSLB4) is compiled and assembled.

5. $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 (IBFTC)

The FORTRAN IV Compiler translates programs written in the FORTRAN IV language and produces input to the Assembler. This input is processed by the Assembler and, if required, by the Loader. The Loader is used for processing and loading when GO, LOGIC, DLOGIC, or MAP is specified in the $IBJOB card. An explanation of these options may be found in the section "IBJOB Card." The object program, which is a result of compiling, assembling, and loading, is composed of generated instructions and subroutines from the Subroutine Library.

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

$IBFTC Card

The format of the $IBFTC card is:

```
1        8          16
$IBFTC   deckname
```

$$\left[\begin{Bmatrix}\underline{\text{NOLIST}}\\ \text{LIST}\\ \text{FULIST}\end{Bmatrix}\right]\left[,\begin{Bmatrix}\underline{\text{NOREF}}\\ \text{REF}\end{Bmatrix}\right]\left[,\begin{Bmatrix}\underline{\text{DECK}}\\ \text{NODECK}\end{Bmatrix}\right]\left[,\begin{Bmatrix}\underline{\text{M90}}\\ \text{M94}\\ \text{M94/2}\end{Bmatrix}\right]\left[,\begin{Bmatrix}\underline{\text{XR3}}\\ \text{XRn}\end{Bmatrix}\right]$$

where deckname identifies the deck that follows. A deck name of six or fewer alphameric 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 blanks.

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

LIST OPTIONS

The list options are:

1. LIST — A listing of the object program, three instructions per line, is generated. Only the relative locations and symbolic information are listed.

2. FULIST — A listing of the object program is generated, one instruction per line. This listing includes generated octal information.

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

If neither LIST, FULIST, nor NOLIST is specified, a listing is not generated.

SYMBOL TABLE OPTIONS

The symbol table options are:

1. REF — A cross-reference table of the symbols used in the object program is generated for listing purposes.
2. NOREF — A cross-reference table is not wanted.

If neither REF nor NOREF is specified, the cross-reference table is not generated.

PUNCH OPTIONS

The punch options are:

1. DECK — The object program deck is written on the System Peripheral Punch Unit for off-line punching.
2. 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.

INSTRUCTION SET OPTIONS

The instruction set options are:

1. M90 — The object program uses only 7090 machine instructions. Any double-precision operations are simulated by system macros, and EVEN pseudo-operations are treated as commentary.
2. M94 — The object program uses 7094 machine instructions.
3. M94/2 — The object program uses 7094 machine instructions, and EVEN pseudo-operations are treated as commentary.

If neither M90, M94, nor M94/2 is specified, it is assumed that the object program uses only 7090 machine instructions.

INDEX REGISTER OPTIONS

The index register options are:

1. XR3 — The object program uses three index registers (1, 2, and 4).
2. XRn — The object program can use up to n index registers, if they are required (n is a number from 4 through 7).

If nothing is specified in this field, it is assumed that the object program uses three index registers.

Sample Deck Formats

Figure 5 shows the control cards that are necessary for the compilation and execution of a single FORTRAN IV language deck.

Figure 6 shows the control cards that are necessary for the compilation and simultaneous execution of two separate FORTRAN IV language decks. When execution begins, control is transferred to the standard entry point of the first deck.

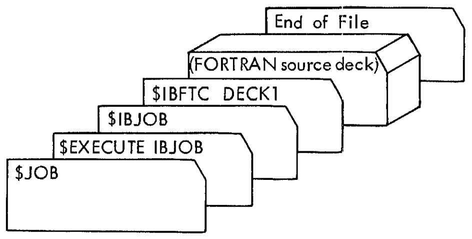


Figure 5. Sample Control Card Deck for One FORTRAN IV Compilation

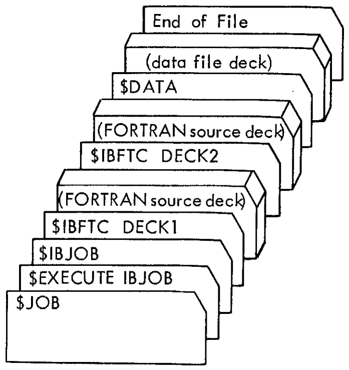


Figure 6. Sample Control Card Deck for Two FORTRAN IV Compilations

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. The reading of the file mark by the IBJOB Processor causes the loading and execution of the object program.

The COBOL Compiler (IBCBC)

The COBOL Compiler translates programs written in the COBOL language and produces input to the Assembler. This input is processed by the Assembler and, if required, by the Loader. The Loader is used for processing and loading when GO, LOGIC, DLOGIC, or MAP is specified in the $IBJOB card. An explanation of these options may be found in the section "$IBJOB Card." The object program, which is a result of compiling, assembling, and loading, is composed of generated instructions and subroutines from the Subroutine Library.

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

$IBCBC Card

The format of the $IBCBC card is:

```
1        8          16
                    ┌ ┌NOLIST ┐┐ ┌   ┌M90  ┐┐
$IBCBC   deckname   │ ┤LIST   ├│ │ , ┤M94  ├│
                    └ └FULIST ┘┘ └   └M94/2┘┘
                                ┌ , ┌DECK  ┐┐ ┌ , ┌NOREF┐┐
                                └   └NODECK┘┘ └   └REF  ┘┘
                                    ┌ , ┌XR3┐┐ ┌ , ┌INLINE┐┐
                                    └   └XR7┘┘ └   └TIGHT ┘┘
                              ┌ , ┌IOEND ┐┐ ┌ , ┌COMSEQ┐┐
                              └   └READON┘┘ └   └BINSEQ┘┘
```

where deckname identifies the deck that follows. A deck name of six or fewer alphameric 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 blanks.

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

LIST OPTIONS

The list options are:

1. LIST — A listing of the object program, three instructions per line, is generated. Only the relative locations and symbolic information are listed.
2. FULIST — A listing of the object program is generated, one instruction per line. This listing includes generated octal information.
3. NOLIST — A listing of the object program is not wanted.

If neither LIST, FULIST, nor NOLIST is specified, a listing is not written on the System Output Unit.

SYMBOL TABLE OPTIONS

The symbol table options are:

1. REF — A sorted dictionary of the source language names and their associated EN (Equivalent Name) numbers and a cross-reference table of the symbols used in the object program are generated for listing purposes.
2. NOREF — A sorted dictionary and a cross-reference table are not wanted.

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

PUNCH OPTIONS

The punch options are:

1. DECK — The object program deck is written on the System Peripheral Punch Unit for off-line punching.
2. 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.

INSTRUCTION SET OPTIONS

The instruction set options are:

1. M90 — The object program uses only 7090 machine instructions.
2. M94 — The object program uses 7094 machine instructions.
3. M94/2 — The object program uses 7094 machine instructions, and EVEN pseudo-operations are treated as commentary.

If neither M90, M94, nor M94/2 is specified, it is assumed that the object program uses only 7090 machine instructions.

INDEX REGISTER OPTIONS

The index register options are:

1. XR3 — The object program uses three index registers (1, 2, and 4).
2. XR7 — The object program can use up to seven index registers, if they are required.

If neither XR3 nor XR7 is specified, the object program uses three index registers.

CODE OPTIONS

The code options are:

1. INLINE — The object program computational and MOVE tasks are optimized for speed.
2. TIGHT — The object program computational and MOVE tasks that are generated are smaller, thereby conserving object-time core storage.

If neither TIGHT nor INLINE is specified, the INLINE specification is assumed.

TAPE ERROR OPTIONS

The tape error options are:

1. IOEND — Errors that occur while reading tape at object time cause an irrecoverable error condition.
2. READON — Any errors that occur while reading tape at object time are ignored. This specification allows high-volume data processing to continue by ignoring low-volume errors.

If neither IOEND nor READON is specified, the IOEND specification is assumed.

COLLATING SEQUENCE OPTIONS

The collating sequence options are:

1. COMSEQ — The object program uses the commercial collating sequence.
2. BINSEQ — The object program uses the binary collating sequence.

If neither COMSEQ nor BINSEQ is specified, the COMSEQ specification is assumed.

$CBEND Card

Every COBOL source deck must be followed immediately by a $CBEND card. The format of this control card is:

```
1
$CBEND
```

Sample Deck Formats

Figure 7 shows the control cards that are necessary for the compilation and execution of a single COBOL language deck.

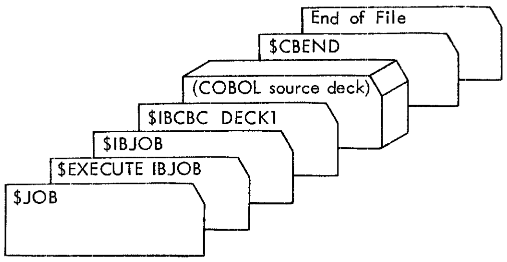


Figure 7. Sample Control Card Deck for One COBOL Compilation

Figure 8 shows the control cards that are necessary for the compilation and simultaneous execution of two separate COBOL language decks. When execution begins, control is transferred to the standard entry point of the first deck.

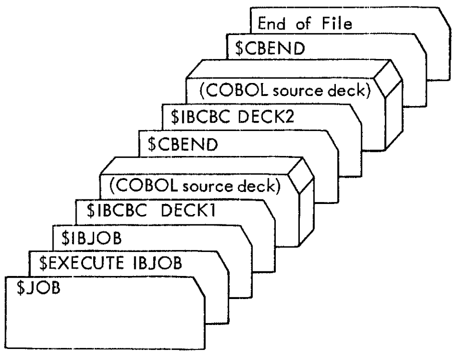


Figure 8. Sample Control Card Deck for Two COBOL Compilations

The Macro Assembly Program (IBMAP)

The Macro Assembly Program (the Assembler) processes programs written in the MAP language and generated MAP programs that are output from the FORTRAN IV and COBOL compilers. 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 $IBJOB card. An explanation of these options may be found in the section "$IBJOB Card." The object program, which is a result of assembling and loading, is composed of machine instructions that are generated by the Assembler and coincide with the MAP mnemonics, 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 "The Subroutine Library (IBLIB)."

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

$IBMAP Card

The format of the $IBMAP card is:

```
1        8         16       [, {LIST   }] [, {REF   }]
$IBMAP   deckname  [count]  [  {NOLIST}] [  {NOREF}]

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

                       [  {RELMOD}] [, {NO (  )}]
                       [, {SYSMOD}] [  {(  ) OK}]
                       [  {ABSMOD}]

                                    [, {NOMFTC}]
                                    [  {MFTC  }]
```

where deckname identifies the deck that follows. A deck name of six or fewer alphameric 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 blanks.

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

CARD COUNT OPTION

The card count option designates the number of symbolic-language cards in the deck. The number cannot exceed five digits. If a number is not specified, a count of 2,000 is assumed.

LIST OPTIONS

The list options are:

1. LIST — A listing of the object program is generated.

2. 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.

SYMBOL TABLE OPTIONS

The symbol table options are:

1. REF — A cross-reference table of the symbols used in the object program is generated for listing purposes.

2. NOREF — A cross-reference table is not wanted.

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

PUNCH OPTIONS

The punch options are:

1. DECK — The object program deck is written on the System Peripheral Punch Unit for off-line punching.

2. NODECK — A punched deck is not wanted.

If neither DECK nor NODECK is specified, the object program is written on the System Peripheral Punch Unit.

INSTRUCTION SET OPTIONS

The instruction set options are:

1. M90 — The object program uses only 7090 machine instructions. Any double-precision operations are simulated, and EVEN pseudo-operations are treated as commentary.

2. M94 — The object program uses 7094 machine instructions.

3. M94/2 — The object program uses 7094 machine instructions, and EVEN pseudo-operations are treated as commentary.

If neither M90, M94, nor M94/2 is specified, it is assumed that the object program uses only 7090 machine instructions.

ASSEMBLY MODE OPTIONS

The assembly mode options are:

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

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

3. 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.

PARENTHESES OPTIONS

The parentheses options are:

1. 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.

2. ( ) OK — Parentheses can be used in MAP symbols.

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

BUILT-IN FUNCTION OPTIONS

The built-in function options are:

1. MFTC — The built-in functions of the FORTRAN IV Compiler are used by the object program. These functions are described in the section "Built-In Functions," in the publication *IBM 7090/7094 Programming Systems FORTRAN IV Language,* Form C28-6274-1.

2. NOMFTC — The built-in functions are not used by the object program.

If neither MFTC nor NOMFTC is specified, it is assumed that the object program does not use the built-in functions.

Sample Deck Formats

Figure 9 shows the control cards that are necessary for the assembly and execution of a single MAP language deck.

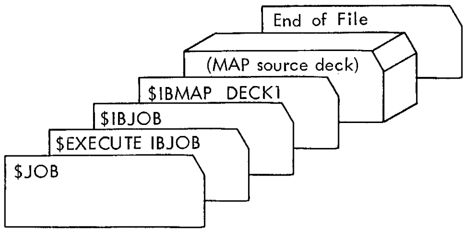


Figure 9. Sample Control Card Deck for One MAP Assembly

Figure 10 shows the control cards that are necessary for the assembly and simultaneous execution of two separate MAP language decks. When execution begins, control is transferred to the standard entry point of the first deck.

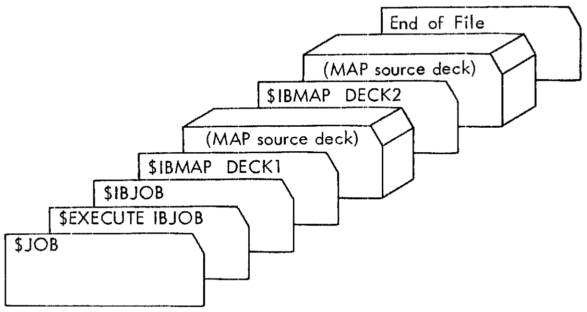


Figure 10. Sample Control Card Deck for Two MAP Assemblies

The Loader (IBLDR)

The Loader processes relocatable binary program decks produced by the Assembler 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 constitute a Processor application when they are executed together. A Processor application can consist of 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 the 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 will perform the desired cross-referencing. External names may be designated using the MAP pseudo-operations CONTRL, ENTRY, and EXTERN; and they may be renamed, replaced, or deleted at load time, using the Loader control cards.

Object Program Files

Object program file control blocks are constructed by the Loader from information contained in $FILE cards and from information supplied by the source program. The user of the MAP language codes 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.

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.

Deck Names: Identify decks and may be used to identify or qualify control section names within a deck.

Control Section Names: 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.

File Names: 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 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 card, $FDICT card, $TEXT card, $CDICT card, and $DKEND card.
3. The deck name may be punched in columns 8-13 of any other Loader control card, but 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, excluding parentheses, commas, slashes, quotation marks, equal signs, and blanks. It 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 that has a relative location assigned within the deck that refers to it.

4. A virtual control section is any control section that has no known origin or length in the deck that refers to it. 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.

7. Explicit inclusion of two control sections with the same name (by using deck name qualification on a $USE card) results in a multiple definition of that section. Consequently, 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. Deletion of an inner level of nesting does not affect the next outer level of nesting.

10. Explicit inclusion of a control section through specification on 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 or CONTRL pseudo-operation should not be the same as the name of the deck that contains it.

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. 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 specifies 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.

OK names + — . $ *

$IBLDR Card

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

```
1       8          16
$IBLDR  deckname  ⎡⎧NOLIBE⎫⎤ ⎡, ⎧TEXT   ⎫⎤
                  ⎣⎩LIBE  ⎭⎦ ⎣  ⎩NOTEXT ⎭⎦
```

where deckname identifies the deck to be processed. A deck name of six or fewer alphameric 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 blanks.

The variable field begins in the column 16. The options of the variable field are described in the following text.

Input Options

The input options are:

1. NOLIBE – The object program is in the current input file.
2. 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.

Text Options

The text options are:

1. TEXT – The text section of the deck is loaded.
2. NOTEXT – The control information in the deck is loaded, but the text section is not loaded.

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, 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 specifications in this card override the file description that appears in the program to be loaded. A $ETC card can be used to extend the variable field of a $FILE card. The format of this control card is:

```
1          16
$FILE      'filename'  [, options, ]
```

where the 'filename' is an alphameric name of 18 or fewer characters that identifies the file; it must be enclosed by quotation marks. The specifications for the options may be entered in any order. Specifications are separated from the file name and from each other by commas.

UNIT ASSIGNMENT OPTIONS

Two units may be specified for each file:

```
[, unit 1, unit 2]
```

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

MOUNTING OPTIONS

The mounting options are:

```
⎡  ⎧MOUNT  ⎫ ⎤
⎢  ⎪DEFER  ⎪ ⎥
⎢  ⎪READY  ⎪ ⎥
⎢ ,⎨  or   ⎬ ⎥
⎢  ⎪MOUNTi ⎪ ⎥
⎢  ⎪DEFERi ⎪ ⎥
⎣  ⎩READYi ⎭ ⎦
```

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.

1. MOUNT – The message is printed before execution, and a stop occurs for the required operator action.
2. DEFER – The operator message and stop are deferred until the file is opened.
3. 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 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 example causes the MOUNT operation for Unit 1 and the DEFER operation for Unit 2:

MOUNT,DEFER2

FILE LIST OPTIONS

The file list options are:

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

1. LIST – The file appears in the operator's mounting instructions.
2. NOLIST – The file does not appear in the operator's mounting instructions.

FILE USAGE OPTIONS

The file usage options are:

$$\left[,\begin{Bmatrix}\text{INPUT}\\\text{OUTPUT}\\\text{INOUT}\\\text{CHECKPOINT}\\\text{or}\\\text{CKPT}\end{Bmatrix}\right]$$

1. INPUT – The file is an input file.
2. OUTPUT – The file is an output file.
3. INOUT – The file may be either an input file or an output file. The object program is responsible for setting the appropriate bits in the file block. Initially, the file is set up as an input file.
4. CHECKPOINT or CGKPT – The file is a checkpoint file.

BLOCK SIZE OPTION

The block size option is:

$$\left[,\begin{Bmatrix}\text{BLOCK}\\\text{BLK}\end{Bmatrix}=\text{XXXX}\right]$$

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

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

ACTIVITY OPTION

The activity option is:

[,ACT = XX]

xx is a number (00-99) that specifies the relative activity of this file with respect to other files. If this field is omitted, the activity is assumed to be 1. This value is used in determining the number of input/output buffers to assign to each buffer pool in the object program.

REEL SWITCHING OPTIONS FOR UNLABELED FILES

The reel switching options for unlabeled files are:

$$\left[,\begin{Bmatrix}\text{ONEREEL}\\\text{MULTIREEL}\\\text{or}\\\text{REELS}\end{Bmatrix}\right]$$

1. ONEREEL – Reel switching should not occur.
2. 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.

REEL SEARCHING OPTIONS FOR LABELED FILES

The reel searching options for labeled files are:

$$\left[,\begin{Bmatrix}\text{NOSEARCH}\\\text{SEARCH}\end{Bmatrix}\right]$$

1. NOSEARCH – If an incorrect label is detected when opening an input file, IOCS stops for operator action.
2. SEARCH – IOCS initiates a multi-reel searching procedure for the file with the desired label.

FILE DENSITY OPTIONS

The file density options are:

$$\left[,\begin{Bmatrix}\text{HIGH}\\\text{LOW}\\\text{200}\\\text{556}\\\text{800}\end{Bmatrix}\right]$$

This field specifies the density at which the file is to be read or written. The file density options are:

1. 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.
2. 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.
3. 200 – The file recording density is 200 cpi.
4. 556 – The file recording density is 556 cpi.
5. 800 – The file recording density is 800 cpi.

If a system unit is assigned to this file, the specifications for these units supersede any of the preceding specifications.

MODE OPTIONS

The mode options are:

$$\left[,\begin{Bmatrix}\text{BCD}\\\text{BIN}\\\text{MXBCD}\\\text{MXBIN}\end{Bmatrix}\right]$$

1. BCD – The file is in BCD mode.
2. BIN – The file is in binary mode.

Form C28-6275-2
Page Revised 2/28/64
by TNL N28-0100

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

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

LABEL DENSITY OPTIONS

The label density options are:

$$\left[,\begin{Bmatrix}\text{SLABEL}\\ \text{HILABEL}\\ \text{LOLABEL}\\ \text{FLABEL}\end{Bmatrix}\right]$$

1. SLABEL – All label operations are performed in the installation standard label specification.

2. HILABEL – All label operations are performed in high density.

3. LOLABEL – All label operations are performed in low density.

4. FLABEL – All 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.

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

BLOCK SEQUENCE OPTIONS

The block sequence options are:

$$\left[,\begin{Bmatrix}\text{NOSEQ}\\ \text{SEQ}\\ \text{or}\\ \text{SEQUENCE}\end{Bmatrix}\right]$$

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

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

CHECK SUM OPTIONS

The check sum options are:

$$\left[,\begin{Bmatrix}\text{NOCKSUM}\\ \text{CKSUM}\end{Bmatrix}\right]$$

1. NOCKSUM – This specifies that the check sum is not written or checked.

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

CHECKPOINT OPTIONS

The checkpoint options are:

$$\left[,\begin{Bmatrix}\text{NOCKPTS}\\ \text{CKPTS}\end{Bmatrix}\right]$$

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

2. CKPTS – Checkpoints are initiated by this file.

CHECKPOINT LOCATION OPTION

The checkpoint location option is:

[,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 file when a reel switch occurs.

FILE CLOSE OPTIONS

The file close options are:

$$\left[,\begin{Bmatrix}\text{SCRATCH}\\ \text{PRINT}\\ \text{PUNCH}\\ \text{HOLD}\end{Bmatrix}\right]$$

1. SCRATCH – The file is rewound upon termination of the application.

2. 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.

3. 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.

4. 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 online 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.

STARTING CYLINDER NUMBER OPTION

The starting cylinder number option is:

$$\left[,\begin{Bmatrix}\text{CYL}\\ \text{CYLINDER}\end{Bmatrix} = \begin{Bmatrix}\text{X}\\ \text{XXX}\end{Bmatrix}\right]$$

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, 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

The cylinder count option is:

Form C28-6275-2
Page Revised 2/28/64
by TNL N28-0100

$\left[, \begin{Bmatrix}\text{CYLCOUNT}\\ \text{CYLCT}\end{Bmatrix} = \begin{Bmatrix}\text{XX}\\ \text{XXX}\end{Bmatrix}\right]$

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

The disk and drum write checking option is:

[, WRITECK]

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

HYPERTAPE REEL SWITCHING OPTIONS

The Hypertape reel switching options are:

$\left[, \begin{Bmatrix}\text{HRFP}\\ \text{HRNFP}\\ \text{HNRFP}\\ \text{HNRNFP}\end{Bmatrix}\right]$

For reel switching to occur, the programmer should specify whether or not the Hypertape is to be rewound and/or file protected.

1. HRFP – Designates Hypertape rewind and file protection.
2. HRNFP – Designates Hypertape rewind, but no file protection.
3. HNRFP – Designates that the Hypertape is not to be rewound, but is to be file protected.
4. HNRNFP – Designates that the Hypertape is not to be rewound or file protected.

$LABEL Card

This control 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. 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'
```

$\left[, \begin{Bmatrix}\text{serial}\\ \text{HA}\end{Bmatrix}\right] \left[, \text{reel}\right] \left[, \begin{Bmatrix}\text{date}\\ \text{days}\end{Bmatrix}\right] \left[, \text{name}\right]$

where filename is an alphameric literal of 18 or fewer characters that identifies the file. It must be enclosed by quotation marks.

SERIAL NUMBER OPTION

The serial number option is:

$\left[, \begin{Bmatrix}\text{serial}\\ \text{HA}\end{Bmatrix}\right]$

Serial – If a tape label is desired, this is 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.

HA – If a labeled disk file or drum file is desired, this field must contain two BCD characters that specify the Home Address-2.

REEL SEQUENCE NUMBER OPTION

The reel sequence number option is:

[reel]

This is a numeric 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 a disk label or drum label is desired, this field must be omitted by using two adjacent commas.

RETENTION CYCLE OPTIONS

The retention cycle options are:

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

1. 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.
2. Date – This is the creation date of the file. The format is Y/D, where Y is a one- or two-digit number indicating the year, and D is a number of three or fewer digits indicating the day of the year. This specification is used for additional label information.

FILE IDENTIFICATION OPTION

The file identification option is:

[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

This control 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
```

$$\left[\begin{Bmatrix}\text{BLOCK}\\\text{BLK}\end{Bmatrix} = \text{XXXX}\right] \left[, \text{BUFCT} = \text{XXX}\right] \left\{, \text{'filename}_1\text{'}, \ldots\right\}$$

BLOCK SIZE OPTION

The block size option is:

$$\left[\begin{Bmatrix}\text{BLOCK}\\\text{BLK}\end{Bmatrix} = \text{XXXX}\right]$$

xxxx is 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

The buffer count option is:

[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. If this field is omitted, the Loader attempts to assign at least two buffers to each file.

FILE NAME SPECIFICATION

The file name specification is:

$$\left\{, \text{'filename'}, \ldots\right\}$$

The remaining data required on the card are the names of the files that are to be included in the pool. Each file name 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.

$GROUP Card

This control 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     [OPNCT = XX] [, BUFCT = XXX]
```

$$\left\{, \text{'filename}_1\text{'}, \ldots\right\}$$

OPEN COUNT OPTION

The open count option is:

[OPNCT = XX]

xx is 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

The buffer count is:

[BUFCT = XXX]

xxx is 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.

FILE NAME SPECIFICATIONS

The file name specification is:

$$\left\{, \text{'filename}_1\text{'}, \ldots\right\}$$

The remaining data required on the card are the names of the files that are to be included in the group. Each file name 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.

$USE Card

This control 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

This control 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

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

This control 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.

$ETC CARD

This control 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 indicates 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 as follows:

```
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 as input/output buffers, the core storage not used by an object program. Storage allocation is not performed 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, a transfer instruction is generated to the first instruction to be executed.
3. IOCS is coded in such a way 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.

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, and its control word is equal to SVN 1, , 1.

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, as follows:

   a. The pool is given one buffer for each file. If available storage does not allow this, 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, a weighing factor is formed from the number of additional buffers that are 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(s) 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 assure 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 (SVNOPNCT, , BUFCT). 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:

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

Unit Assignment Specifications

DESIGNATION	EXPLANATION
blank	Use any available unit.
M	Use any available model M 729 Magnetic Tape Unit, where M is either a II, IV, V, or VI.

X	Use any available unit on channel X.
P	Use any available unit on channel P.
X(k)	Use the kth available 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	Use any available unit on channel I. This specification can be used for input and output units.
IM	Use any available 729 Magnetic Tape Unit, model M, on channel I. This specification can be used only for output units.
I(k)	Use the kth available 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.
IR	Use any available unit on channel I, and release the unit from reserve status after the application has been completed.
I(k)R	Use the kth available 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 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 Switches.
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.
OU, OU1, OU2	Use the System Output Unit.
PP, PP1, PP2	Use the System Peripheral Punch Unit.
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 units 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 could 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 re-

maining intersystem units are now assigned by repeating the same process, i.e., finding the intersystem channel with the greatest requirement 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. Files with model specifications only 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 could 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 a block size of 464 words.

The Overlay Structure

Figure 11 is an example of the structure of an overlay application. In Figure 11, 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.

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.

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.

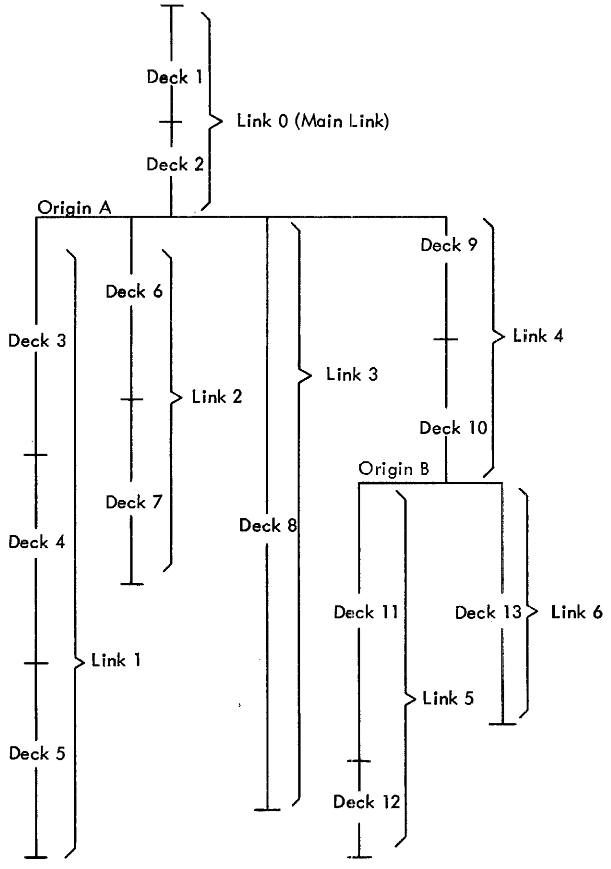


Figure 11. Example of Overlay Structure

FORTRAN and COBOL statements that are translated into a CALL pseudo-operation that refers to another deck can be used to cause overlay.

THE CALL STATEMENT

The primary rule regarding CALL statements that cause links to be loaded is the following:

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 towards 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 SIBJOB 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 11, the following examples may be given:

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

2. A call from Deck 2 to Deck 12. – This call 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. – This call is permitted. This call is in the chain of links toward the main link.

4. A call from Deck 13 to Deck 3. – This call 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. This call is not permitted. The 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 has been 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 12 illustrates how the overlay structure in Figure 11 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.

Library subroutines in the main link are loaded following the input decks that constitute the main link. Library subroutines can be included in deeper links by using a $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, is assigned following the Library subroutines.

In Figure 12, the possible configuration of links in core storage at any one time is:

Links 0 (main link) only
Links 0 and 1
Links 0 and 2
Links 0 and 3
Links 0 and 4
Links 0, 4, and 5
Links 0, 4, and 6

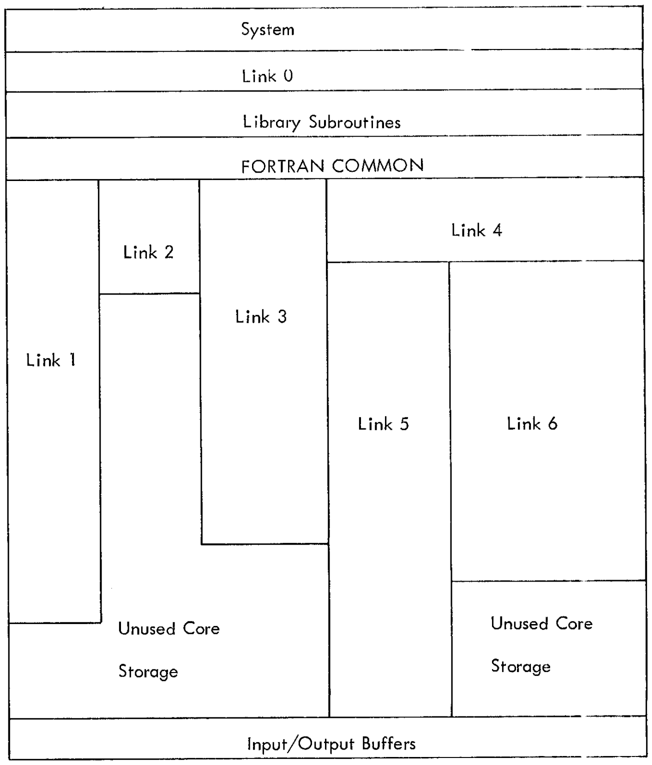


Figure 12. Overlay Core Storage Allocation

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 that 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 this specifies more than one main link. The format of the $ORIGIN card is:

```
1          16
$ORIGIN    logical  ┌, absolute┐ ┌, {SYSUT2}┐
           origin   └  origin  ┘ └  {SYSxxx}┘
                                  ┌, {NOREW}┐
                                  └  {REW  }┘
```

This control 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.

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:

Logical Origin: This field must be the first subfield in the variable field, and it must be specified. 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, periods, and blanks.

Absolute Origin Option: The absolute origin option is:

```
┌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, an alphabetic O must precede the number. This field is used only if a program requires 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 links proceeding from it, if their $ORIGIN cards do not specify an absolute origin.

Unit Specification Options: The unit specification options are:

```
┌, {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)	

If the System Library Unit SYSLB2, SYSLB3, or SYSLB4 is specified and is also used for System residence, 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: The rewind options are:

```
┌{NOREW}┐
└{REW  }┘
```

1. NOREW – The input/output unit containing the link is not to be rewound after the link is loaded.
2. REW – The unit is to be rewound.

If this field is omitted, the unit is not rewound.

$INCLUDE CARD

The format of the $INCLUDE card is:

```
1          16
$INCLUDE   {Deckname}, . . .
           {Exname  }
```

This control 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 subfields of the variable field contain alphameric literals which specify either a deck name (usually a library subroutine) or a real control section name of non-zero 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.

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 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 13 illustrates how a deck would be set up to produce the program structure given in Figure 14.

In Figure 13, 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.

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 which contains Deck 1, the following sequence could be used:

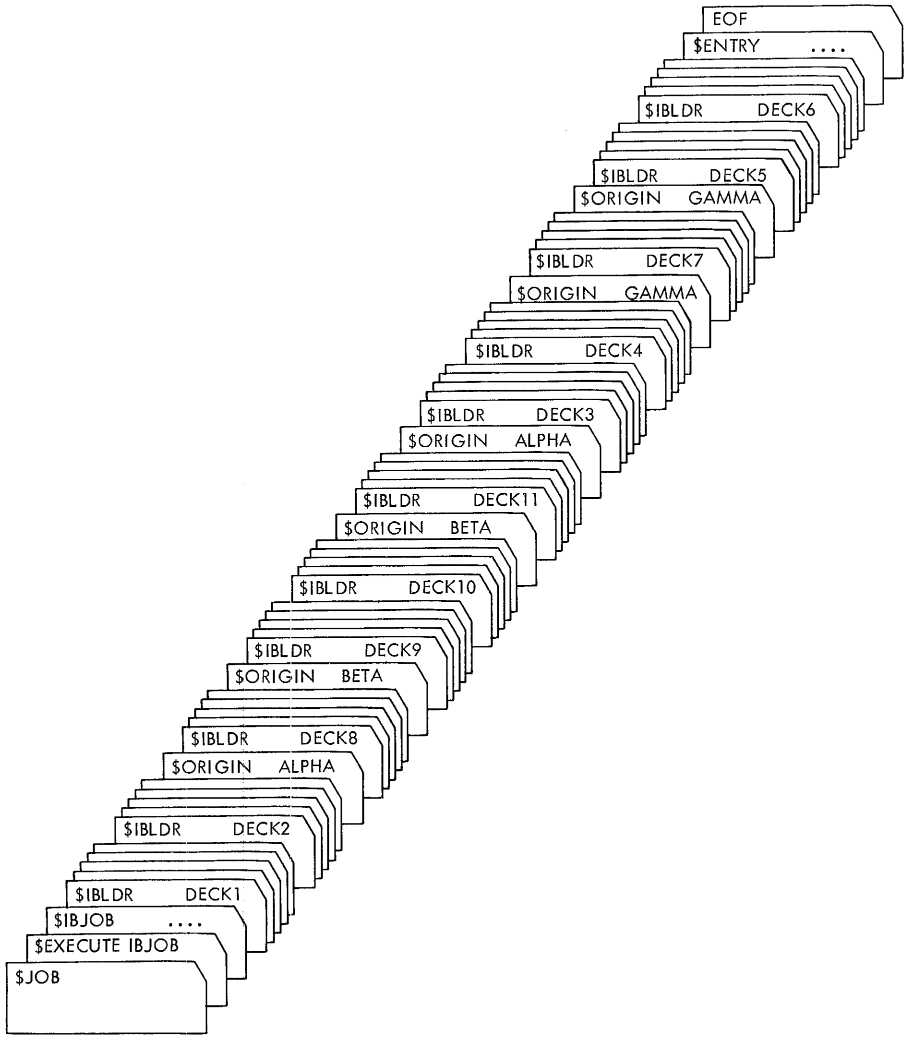


Figure 13. Sample Control Card Deck

```
1            16
$ORIGIN      ALPHA
$INCLUDE     FLOG, XYZ
Deck 1
  .
  .
  .
```

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 which cannot be used efficiently during execution.

For example, in the following sequence:

```
1                         16
$ORIGIN                   A,SYSUT3      (Link A)
$INCLUDE                  XYZ
  .
  .
  .
$ORIGIN                   B,SYSUT3      (Link B)
$IBLDR                    DECK 1
$IBLDR                    DECK 2
                          (contains section XYZ)
```

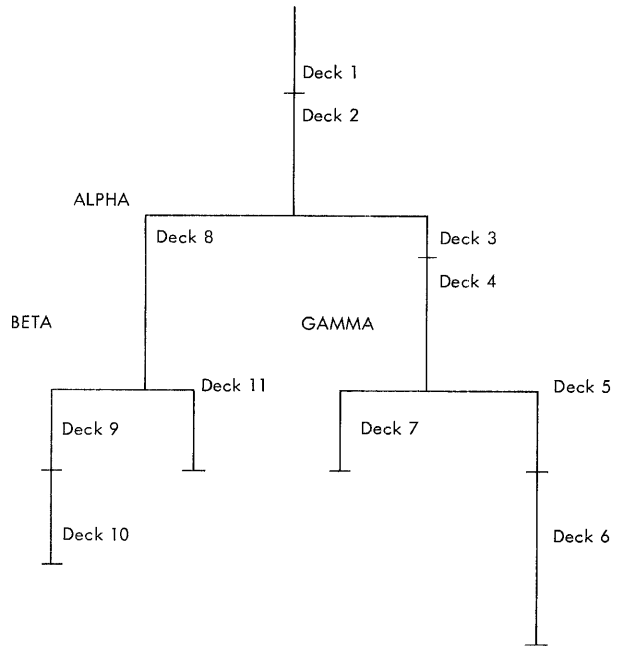


Figure 14. Overlay Program Structure

the instructions or data in section XYZ which are to become part of link A will not be 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 will be isolated from the main portion of link A. If SYSUT3 is a tape file, some tape will have 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 which 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.

The 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 generates 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 was set to TXL and a transfer would now be 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 properly indicate the load status of the links.

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

The Subroutine Library (IBLIB)

The Subroutine Library contains a collection of relocatable subroutines for system and programmer use. These subroutines are available to the programmer through the Loader, which incorporates them, as required, into the object program at load time.

Subroutines may be added to or deleted from the Subroutine Library by using the Librarian. Information regarding modification of the Subroutine Library may be found in the section "The Librarian." All subroutines included in the Subroutine Library must be assembled by the Assembler.

The Subroutine Library is composed of system subroutines, COBOL subroutines, and FORTRAN IV subroutines. The FORTRAN section of the Subroutine Library is subdivided into the FORTRAN Mathematics Library, the FORTRAN Input/Output Library, and the FORTRAN Utility Library.

The FORTRAN mathematics subroutines and the FORTRAN utility subroutines for testing machine indicators and for recording the status of the console and selected portions of core storage are described in this section.

The other subroutines are not of general concern to the application's programmer and, for this reason, are described in the section "Subroutine Library Information."

FORTRAN Mathematics Library

The FORTRAN Mathematics Library consists of three types of subroutines: single-precision, double-precision, and complex. These subroutines are described in this section. Subroutines that are marked with an asterisk have entry points that cannot be used in the FORTRAN IV language. These entry points can, however, be used in a MAP language program.

Single-Precision Subroutines

The single-precision subroutines are described in the following text.

THE FXP1 SUBROUTINE

A call to this subroutine is compiled for a source program exponential term such as I**J.

Entry Point: .XP1. is the entry point to this subroutine for an exponential expression with a fixed-point base and a fixed-point exponent. The output is a fixed-point number in the AC.

MAP Call: The MAP language call to this subroutine is:

```
          CALL    .XP1.(I,J)
```

THE FXP2 SUBROUTINE

A call to this subroutine is compiled for a source program exponential term such as A**J.

Entry Point: .XP2. is the entry point to this subroutine for an exponential expression with a floating-point base and a fixed-point exponent. The output is a normalized floating-point number in the AC.

MAP Call: The MAP language call to this subroutine is:

```
          CALL    .XP2.(A,J)
```

THE FXP3 SUBROUTINE

A call to this subroutine is compiled for a source program exponential term such as A**B.

Entry Point: .XP3. is the entry point to this subroutine for an exponential expression with a floating-point base and a floating-point exponent. The output is a normalized floating-point number in the AC.

MAP Call: The MAP language call to this subroutine is:

```
          CALL    .XP3.(A,B)
```

THE FXPF SUBROUTINE

A call to this subroutine is compiled for a source program expression such as Y = EXP(A). It computes $_eA$, the natural antilogarithm of the number A.

Entry Point: EXP is the entry point to this subroutine for natural antilogarithm expressions. The output is a normalized floating-point number in the AC.

MAP Call: The MAP language call to this subroutine is:

```
          CALL    EXP(A)
```

THE FLOG SUBROUTINE

A call to this subroutine is compiled for source program expressions such as Y = ALOG(A) and Y = ALOG10(A). It computes the natural logarithm or common logarithm of the number A.

Entry Points: The following entry points are contained in the FLOG subroutine:

1. ALOG is the entry point for the computation of the natural logarithm of the number A. The output is a normalized floating-point number in the AC.

The MAP language call is:

```
          CALL    ALOG(A)
```

2. ALOG10 is the entry point for the computation of the common logarithm of the number A. The output is a normalized floating-point number in the AC.

The MAP language call is:

```
          CALL    ALOG10(A)
```

THE FATN SUBROUTINE

A call to this subroutine is compiled for source program expressions such as Y = ATAN(A) and Y = ATAN2(A,B). It computes the arctangent, in radians, of one or two arguments.

Entry Points: The following entry points are contained in the FATN subroutine:

1. ATAN is the entry point for the computations of the arctangent argument A. The output is a normalized floating-point number in the AC.

The MAP language call is:

```
        CALL    ATAN(A)
```

2. ATAN2 is the entry point for the computation of the arctangent of A/B. The output is a normalized floating-point number in the AC.

The MAP language call is:

```
        CALL    ATAN2(A,B)
```

THE FSCN SUBROUTINE

A call to this subroutine is compiled for source language expressions such as Y = SIN(A) and Y = COS(A). It computes the sine or cosine of an angle expressed in radians.

Entry Points: The following entry points are contained in the FSCN subroutine.

1. SIN is the entry point for the computation of the sine of argument A. The output is a normalized floating-point number in the AC.

The MAP language call is:

```
        CALL    SIN(A)
```

2. COS is the entry point for the computation of the cosine of argument A. The output is a normalized floating-point number in the AC.

The MAP language call is:

```
        CALL    COS(A)
```

THE FTNH SUBROUTINE

A call to this subroutine is compiled for a source language expression such as Y = TANH(A). It computes the hyperbolic tangent of the number A.

Entry Point: TANH is the entry point for the computation of the hyperbolic tangent of argument A. The output is a normalized floating-point number in the AC.

MAP Call: The MAP language call to this subroutine is:

```
        CALL    TANH(A)
```

THE FSQR SUBROUTINE

A call to this subroutine is compiled for a source language expression such as Y = SQRT(A). It computes the positive square root of the number A.

Entry Point: SQRT is the entry point for the computation of the square root of argument A. The output is a normalized floating-point number in the AC.

MAP Call: The MAP language call to this subroutine is:

```
        CALL    SQRT(A)
```

Double-Precision Subroutines

The double-precision subroutines are described in the following text.

THE FDMD SUBROUTINE

A call to this subroutine is compiled for a source language expression such as Y = DMOD(D,E). It computes D modulo E (defined as D – [D/E] *E, where the brackets indicate that only the integer portion of the expression within them is to be used in evaluating the equation).

Entry Point: DMOD is the entry point for the computation of D modulo E. The output is a normalized double-precision floating-point number in the AC and MQ.

MAP Call: The MAP language call to this subroutine is:

```
        CALL    DMOD(D,E)
```

THE FDX1 SUBROUTINE

A call to this subroutine is compiled for source language expressions such as D**I and C**I.

Entry Points: The following entry points are contained in the FDX1 subroutine:

1. .DXP1. is the entry point to this subroutine for an exponential expression with a double-precision floating-point base and a fixed-point exponent. The output is a normalized double-precision floating-point number in the AC and MQ.

The MAP language call is:

```
        CALL    .DXP1.(D,I)
```

2. .CXP1. is the entry point to this subroutine for an exponential expression with a complex base and a fixed-point exponent. The output is a complex number with the real portion in the AC and the imaginary portion in the MQ.

The MAP language call is:

```
        CALL    .CXP1.(C,I)
```

THE FDX2 SUBROUTINE

A call to this subroutine is compiled for source language expressions such as D**A and D**E.

Entry Point: .DXP2. is the entry point to this subroutine for the following exponential expressions:

1. An exponential expression with a double-precision floating-point base and a single-precision floating-point exponent. The output is a normalized double-precision floating-point number in the AC and MQ.

The MAP language call is:

```
          CALL    .DXP2.(D,A)
```

2. An exponential expression with a double-precision floating-point base and a double-precision floating-point exponent. The output is a normalized double-precision floating-point number in the AC and MQ.

The MAP language call is:

```
          CALL    .DXP2.(D,E)
```

THE FDXP SUBROUTINE

A call to this subroutine is compiled for a source language expression such as Y = DEXP(D).

Entry Point: DEXP is the entry point for the computation of the natural antilogarithm of the double-precision number D. The output is a normalized double-precision floating-point number in the AC and MQ.

MAP Call: The MAP language call to this subroutine is:

```
          CALL    DEXP(D)
```

THE FDLG SUBROUTINE

A call to this subroutine is compiled for source language expressions such as Y = DLOG(D) and Y = DLOG10(D). It computes the natural logarithm or common logarithm of the double-precision argument D.

Entry Points: The following entry points are contained in the FDLG subroutine:

1. DLOG is the entry point for the computation of the natural logarithm of the number D. The output is a normalized double-precision floating-point number in the AC and MQ.

The MAP language call is:

```
          CALL    DLOG(D)
```

2. DLOG10 is the entry point for the computation of the common logarithm of the number D. The output is a normalized double-precision floating-point number in the AC and MQ.

The MAP language call is:

```
          CALL    DLOG10(D)
```

THE FDSQ SUBROUTINE

A call to this subroutine is compiled for source language expression such as Y = DSQRT(D). It computes the positive square root of the double-precision argument D.

Entry Point: DSQRT is the entry point for the computation of the square root of the number D. The output is a normalized double-precision floating-point number in the AC and MQ.

MAP Call: The MAP language call to this subroutine is:

```
          CALL    DSQRT(D)
```

THE FDSC SUBROUTINE

A call to this subroutine is compiled for source language expressions such as Y = DSIN(D) and Y = DCOS(D). It computes the sine or cosine of an angle expressed in radians.

Entry Points: The following entry points are contained in the FDSC subroutine:

1. DSIN is the entry point for the computation of the sine of the double-precision argument D. The output is a normalized double-precision floating-point number in the AC and MQ.

The MAP language call is:

```
          CALL    DSIN(D)
```

2. DCOS is the entry point for the computation of the cosine of the double-precision argument D. The output is a normalized double-precision floating-point number in the AC and MQ.

The MAP language call is:

```
          CALL    DCOS(D)
```

THE FDAT SUBROUTINE

A call to this subroutine is compiled for source language expressions such as Y = DATAN(D) and Y = DATAN2(D,E). It computes the arctangent, in radians, of one or two double-precision arguments.

Entry Points: The following entry points are contained in the FDAT subroutine:

1. DATAN is the entry point for the computation of the arctangent of argument D. The output is a normalized double-precision floating number in the AC and MQ.

The MAP language call is:

```
          CALL    DATAN(D)
```

2. DATAN2 is the entry point for the computation of the arctangent of D/E. The output is a normalized double-precision floating-point number in the AC and MQ.

The MAP language call is:

```
          CALL    DATAN2(D,E)
```

Complex Subroutines

The complex subroutines are described in the following text.

THE FCAS SUBROUTINE

A call to this subroutine is compiled for source language expressions such as C*F and C/F. It performs multiplication and division of complex numbers.

Entry Points: The following entry points are contained in the FCAS subroutine:

1. .CFMP. is the entry for complex multiplication. The output is a complex number in the AC and MQ.

The MAP language call is:

CALL .CFMP.(C,F)

2. .CFDP. is the entry point for complex division. The output is a complex number in the AC and MQ.

The MAP language call is:

CALL .CFDP.(C,F)

THE FCAB SUBROUTINE

A call to this subroutine is compiled for a source language expression such as Y = CABS(C).

Entry Point: CABS is the entry point for the computation of the absolute value of the argument C. The output is a complex number in the AC and MQ.

MAP Call: The MAP language call to this subroutine is:

CALL CABS(C)

THE FCXP SUBROUTINE

A call to this subroutine is compiled for a source language expression such as Y = CEXP(C).

Entry Point: CEXP is the entry point to this subroutine for the computation of the natural antilogarithm of the argument C. The output is a complex number in the AC and MQ.

MAP Call: The MAP language call to this subroutine is:

CALL CEXP(C)

THE FCLG SUBROUTINE

A call to this subroutine is compiled for a source language expression such as Y = CLOG(C).

Entry Point: CLOG is the entry point to this subroutine for the computation of the natural logarithm of the argument C. The output is a complex number in the AC and MQ.

MAP Call: The MAP language call to this subroutine is:

CALL CLOG(C)

THE FCSQ SUBROUTINE

A call to this subroutine is compiled for a source language expression such as Y = FCSQ(C).

Entry Point: CSQRT is the entry point to this subroutine for the computation of the principal square root of the argument C. The output is a complex number in the AC and MQ.

MAP Call: The MAP language call to this subroutine is:

CALL CSQRT(C)

THE FCSC SUBROUTINE

A call to this subroutine is compiled for source language expressions such as Y = CSIN(C) and Y = CCOS(C).

Entry Points: The following entry points are contained in the FCSC subroutine:

1. CSIN is the entry point for the computation of the sine of argument C. The output is a complex number in the AC and MQ.

The MAP language call is:

CALL CSIN(C)

2. CCOS is the entry point for the computation of the cosine of the argument C. The output is a complex number in the AC and MQ.

The MAP language call is:

CALL CCOS(C)

FORTRAN Utility Library

The FORTRAN utility subroutines for testing machine indicators and for recording the status of the console and selected portions of core storage are described in this section. 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. An I is used to specify any integer expression, and a J is used to specify any integer variable in the descriptions of the subroutines.

THE FSLITE SUBROUTINE

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

1. 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.

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

THE FSSWTH SUBROUTINE

This 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 down, J is set to 1; if it was up, J is set to 2.

THE FOVERF SUBROUTINE

This 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.

THE FDVCHK SUBROUTINE

This 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 following subroutine can be referred to by CALL statements in either the FORTRAN IV or MAP language.

THE FDMP SUBROUTINE

This subroutine causes the dumping of selected portions of core storage. The limits of the dump and the format of the dump are specified following the entry point specifications.

Entry Points: DUMP is the entry point for a post-mortem dump. After the dump has been taken, core storage is restored and control is returned to the subroutine .LXCON.

PDUMP is the entry point for a snapshot of selected portions of core storage. After the dump has been taken, core storage is restored and execution of the program resumes.

CALL Statements: The following CALL statements are used in both the FORTRAN IV and MAP languages:

CALL DUMP(A_1, B_1, F_1, . . . A_i, B_i, F_i)

CALL PDUMP(A_2, B_1, F_1, . . . A_i, B_i, F_i)

where A and B are variable data names and F is an integer indicating the dump format.

The integers that specify the dump format are:

0 — octal
1 — floating-point
2 — integer
3 — octal and mnemonics

If an integer is not specified, the dump format is octal. When more than one set of arguments is specified, the format for the last set can be omitted, e.g.:

```
        CALL    DUMP(A,B,1,C,D,3,E,F).
```

If no arguments are specified following the entry point, all of core storage is dumped in octal.

FORTRAN Files

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, as shown in Figure 15.

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

Figure 15. Correspondence Between FORTRAN Logical Units and System Files

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    UNITn
UNITn    FILE   specifications
```

where n 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 subroutine.

Variable Units

Any FORTRAN IV source program input/output statement that refers to a variable unit causes the FVIO subroutine and all File routines to be loaded with the object program. The following is an example of such an input/output statement:

```
        WRITE    (I,10)A
```

In this example, the FORTRAN input/output logical unit I varies during execution of the program. The FVIO subroutine 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.

Error Procedures

When an object program error is detected by the Subroutine Library, a message stating the error condition is written on the System Output Unit. The execution of the mathematical subroutines can be resumed by using an optional exit, even though an error condition has occurred. A complete description of the error conditions and the optional exits is contained in the section "Subroutine Library Information."

Systems Programmer's Information

IBJOB Processor Information

This section contains information which is mainly intended for the systems programmer. It is not necessary for the programmer to have an understanding of this material in order to use the IBJOB Processor, but such an understanding may be useful.

Programming Analysis Aids

The IBJOB Processor incorporates the following facilities to assist the systems programmer:

1. A standard storage location (SYSFAZ). This location indicates the name of the phase currently in core storage.

2. A standard storage location (SYSLOC). This is the location in which the location counter is stored on each call to a subprogram. This makes it easier to determine the section of coding that is responsible if an unlisted stop or endless-loop condition occurs.

The IBJOB Processor uses system macros in place of the standard CALL, SAVE, and RETURN pseudo-operations. The linkage produced by the macro

```
label1    SAVE    n, [I]
```

is as follows:

```
label1    AXT     *+n+4[+1],0
          SXA     *+n+3[+1],4
          SXA     *+n+3[+1],2
          SXA     *+n+3[+1],1
          [STI    .LDIR. +2]
          CLA     SYSLOC
          STA     .LDIR.
          TRA     *+n+2[+1]
          AXT     **,4
          AXT     **,2
          AXT     **,1
          LDI     .LDIR. +2
          XEC     .LDIR.
```

The information within brackets is used only when an I is specified in the SAVE macro.

A remote sequence of coding (called the linkage director) consists of the following coding:

```
.LDIR.    TRA*    **
          BCI     1,label1
          BSS     1
```

The linkage produced by the macro

```
label2    CALL    name,arg1,arg2, . . . , argi
```

is as follows:

```
label2    STL     SYSLOC
          TXH     *+3+n, , n
          TXI     name, , .LDIR.
          BCI     1,label2
          PZE     arg1
          PZE     arg2
           .
           .
           .
          PZE     argi
          TRA*    label1
```

The standard linkage produced by the Macro Assembly Program from the pseudo-operations CALL, SAVE, and RETURN is used with object programs. This is described in the publication *IBM 7090/7094 Programming Systems: Macro Assembly Program (MAP) Language,* Form C28-6311.

SYSLOC and similar system symbols designate words in a communication region. If a system symbol that is defined in a Library subroutine is referred to in an object program, the Loader assigns addresses for system references and thus permits instructions which cannot use indirect addresses to refer to these communication words.

3. Correct and up-to-date comments on systems listings.

4. Well-defined and appropriately labeled storage areas for variable information.

5. Heading lines and extra spacing at appropriate places to facilitate use of listings.

6. Standard stop options for error conditions.

7. If the Overlay feature is being used, checking is performed to prevent a calling sequence program from being overlaid by the program it calls.

Additional Index Register Mode

The IBJOB Processor operates in the additional index register mode at all times. It enters this mode when it receives control from the System Monitor. The IBJOB Processor leaves the additional index register mode before returning control to the System Monitor.

The additional index register mode is also the normal mode for the execution of object programs under the control of the IBJOB Processor. Compiled programs do not use the multiple tag mode. If a program using the multiple tag mode, coded by the programmer in the MAP language, is called by a compiled program, the MAP language program must not have destroyed any index registers or the additional index register mode when control is returned to the calling program. If a MAP language program that uses the multiple tag mode calls a compiled program, the compiled program enters the additional index register mode automatically. Thus, the instruction in the MAP language program, immediately following the call, should be to re-enter 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.

Processor Monitor Information

The monitor of the IBJOB Processor consists of the following components:

1. Job Control. This section of the Processor Monitor receives control from and returns control to the System Monitor, and calls the Loader, if necessary, for a particular Processor application.
2. Process Control. This section of the Processor Monitor receives control from Job Control. It determines the phases of the IBJOB Processor that are required for the application, and calls them into core storage in the proper order.

Job Control

Job Control contains routines to call Process Control and the Loader when an object program is to be loaded.

When Job Control receives control from the System Monitor, it initializes the System Unit Position Table and calls Process Control. When Process Control is finished, it transmits a word to Job Control. By testing this word, Job Control determines whether to transfer to location SYSRET in the Nucleus (IBNUC) or to call the Loader into core storage. If the object program is loaded, but not executed, the Loader returns control to Job Control, which again calls Process Control.

Process Control

Process Control manipulates the Input/Output Editor (the program that performs all input/output for the Processor) by means of its entry point (IOEDIT), and contains a control card search, an option scan, error procedures, and routines to call subsystems.

Initialization

After being called by Job Control, Process Control checks to make sure that necessary input/output units have been assigned by scanning the System Unit Function Table. Files to be used by Process Control are then attached and opened, and the Input/Output Editor is initialized. Initialization of the Input/Output Editor is accomplished by calling IOEDIT and giving the locations of file control blocks and a control location containing flags. 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 use system units for the following files:

System Input Unit (SYSIN1): IBJOB Processor input.

System Output Unit (SYSOU1): IBJOB Processor listing output.

System Peripheral Punch (SYSPP1): IBJOB Processor punched output.

System Utility Unit 1 (SYSUT1): Unused by Process Control and the Input/Output Editor.

System Utility Unit 2 (SYSUT2): IBJOB Processor load file.

System Utility Unit 3 (SYSUT3): Unused by Process Control and the Input/Output Editor.

System Utility Unit 4 (SYSUT4): FORTRAN Compiler or COBOL Compiler output and Macro Assembly Program input from the FORTRAN Compiler or the COBOL Compiler.

After initialization, Process Control starts its control card search. Control is relinquished to a compiler or to the Marco Assembly Program when a $IBFTC, $IBCBC, or IBMAP card is encountered. Control is transferred to the Loader when NOSOURCE is specified on the $IBJOB card or when a file mark or $DATA card has been read and the program must be loaded. Control is returned to the System Monitor after execution of an object program or when a $IBSYS, $JOB, $EXECUTE, or $STOP card is read.

Input/Output Editor

The Input/Output Editor is the section of the Processor Monitor that regulates the input/output functions of the IBJOB Processor. All system input/output is performed by the Input/Output Editor, which provides a line of input or accepts punch or listing output upon request. The Input/Output Editor writes the compiler output and reads it for the Assembler and then writes the load file and reads it for the

Loader. Intermediate input/output and on-line printing are not performed through the Input/Output Editor.

Initialization

An entry point (IOEDIT) is used to transfer to the Input/Output Editor, causing initialization of control information, buffer truncation, and release. All information taken from $IEDIT, $OEDIT, or $IBJOB cards that affects the Input/Output Editor is transmitted through this entry point. Process Control calls to IOEDIT to truncate and release output buffers and to transmit control information.

Input

The Input Editor (JOBIN) is the input phase of the Input/Output Editor. It 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 a compiler), the load file input to the Loader (Assembler output), an input file on an optional unit specified on a $IEDIT card, or the Alter deck that is 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.

After being called with 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 with the accumulator negative and a 1 in the address portion of the accumulator. If an end-of-file condition is sensed, the Input Editor returns control to the calling program with the accumulator negative and a 0 in the address portion of the accumulator.

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.

Listing Output

The Output Editor (JOBOU) is the listing phase of the Input/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, listing output from all components of the IBJOB Processor except the Processor Monitor is placed on the alternate unit. The Output Editor places the listing output of the Processor Monitor on the System Output Unit, but reverts to 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 ejects pages and inserts page headings when necessary. The page heading is given to the Output Editor by Process Control through IOEDIT.

The Output Editor generates two types of output. The status of the word at location TYPOU determines the type of output to be 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 location TYPOU is 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. This word contains $(7600000000xx)_8$, where xx is the number of records (in BCD mode) 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 mode) contained in the record.

A call to the Output Editor initiates a new line when this 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.

Other Output

The Punch Editor (JOBPP) 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 as follows:

Punch file
Load file
Compiler output file

All Punch Editor output files are binary. 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 placed in the file as BCD and are written in binary. $IBJOB control cards, with NOSOURCE specified starting in column 16 and the date specified starting in column 61, are punched when they are read.

If the Punch Editor determines that output is not from a compiler, the card image is placed in the punch file and is recorded properly if it is BCD. The card is also placed in the load file if the proper bit in the control word 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.

Action Routine

The Action 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 and their formats are:

Action Table: This table is used to identify the Action label supplied by the caller. Each entry consists of two words, the first of which is:

```
          BCI     1,LABEL
```

The second word is either:

```
          PZE     a
```

where a is the location of the Action list, or:

```
          MZE     n
```

in which case, the second word is the Action list, and n is as specified for the Action list. The list is limited to one word in this case, and the MON prefix is also allowed.

Action List: This list is located at a, specified in the Action table, or follows the first word of an Action table entry. It 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 an entry in the unit position table and pfx is one of the following:

1. PZE or MZE, which causes positioning of the system unit to the indicated position.
2. PON or MON, which causes positioning of the system unit to the indicated position and reading of the record at that position.

Unit Position Table: This table consists of a one-word entry for each record on the system unit(s). An entry is designated in the address of an Action list entry. This table can have two formats, depending on whether the system is on one or more magnetic tape units or on disk storage.

1. Tape(s) — The format for an entry is:

```
          pfx     rccnt, , flcnt
```

where pfx is the library tape number on which the record exists, and rccnt and flcnt indicate the position of the record on that unit. If pfx is PON, indicating System Library Unit 1 (SYSLB1), the file position is relative to the position of System Library Unit 1 when IBJOB Processor operation begins. The IBJOB Processor converts this relative position table to actual positions during its initialization. If pfx is other than PON, flcnt must be the actual file position.

2. Disk Storage and Drum Storage — 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 Supervisor (IBSUP).

System Record Format

Each system record must be preceded by the following instruction sequence:

```
          IOCP    SYSFAZ, , 1
          BCI     1, RECNM
```

This may be followed by a command to read a transfer to an entry point into the appropriate transfer location in Job Control or elsewhere.

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

Pre-Positioning Feature of the Processor Monitor

The Action routine in the Processor Monitor is used to pre-position a library tape, when this is possible. If the system is assembled for disk, drum, or Hypertape, the pre-positioning feature is inoperative.

Using One Library Unit for the IBJOB Processor

If the System tape is on an IBM 729 Magnetic Tape Unit, the pre-positioning feature performs the following actions. After the last record of the Assembler is read into core storage, the next control card is read. If this control card is a $IBFCT, $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 System tape is 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 is read into core storage.

Using Multiple Library Units for the IBJOB Processor

Several configurations are possible when multiple library units are used to split the IBJOB Processor. These configurations are as follows:

Two IBM 729 Magnetic Tape Units: Full use of the pre-positioning feature can be made using this configuration. Rules governing this configuration are:

1. Both units must be on the same channel.
2. All records of a component of the IBJOB Processor must be on the same tape.
3. If more than two units are used, the pre-positioning 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 initiated for the tape that contains the Processor Monitor and the other System tape is rewound. The backspace file operation is not initiated if neither of the compilers nor the Assembler is on the tape that contains the Processor Monitor.

After the last record of the Loader is read into core storage, both System tapes are rewound.

System on Disk Storage or Drum Storage and the Subroutine Library on Tape: If the System is on disk storage or drum storage, only the files for the Subroutine Library may be on tape. The tape for the Subroutine Library must be on an IBM 729 Magnetic Tape Unit.

After the last record of the Loader is read into core storage, the Library tape is rewound.

Control Card Search

The Control Card Search routine (CCS) calls the Input 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. 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.

Cards that do not have a dollar sign in column 1 are not sent to the Loader by the Processor Monitor. 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 sent to the Loader.

The occurrence of a $EDIT card, indicating that a Subroutine Library edit is to be performed, causes the Processor Monitor to call the Loader without regard for the requirements of the current application.

Mandatory Card Controlling an Application

$IBJOB CARD

This must be the first card of every IBJOB Processor application. If the GO, MAP, LOGIC, or DLOGIC specification is present on this control card, a load file is started and the $IBJOB card is the first card placed in it, unless the NOSOURCE specification is used. Any cards that do not contain a dollar sign in column 1 and are not recognized by Process Control, and any object decks and Assembler output, are placed in the load file. If the NOSOURCE specification is present, signifying that there is no compilation or assembly in this application, Process Control returns to Job Control, which calls the Loader to load from the System Input Unit. If the GO, MAP, LOGIC, or DLOGIC specifications are not present, no loading is to be performed, so no load file is created.

Component Control Cards

$IBFTC CARD

Process Control sets the Input/Output Editor controls for FORTRAN IV compilation and calls the FORTRAN IV Compiler (IBFTC). The FORTRAN IV Compiler must call the Input Editor to get this control card, which must immediately precede every FORTRAN IV source deck. Process Control automatically calls the Assembler to assemble FORTRAN IV Compiler output, if the error level permits assembly.

$IBCBC CARD

Process Control sets the Input/Output Editor controls for a COBOL compilation and calls the COBOL Compiler (IBCBC). The COBOL Compiler must call the Input Editor to get this card, which must immediately precede every COBOL source deck. 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 Macro Assembly Program (IBMAP). The Assembler must call the Input Editor to get this control card, which must immediately precede every symbolic deck.

$IBLDR CARD

Process Control adds this control card and either the object deck following this control card or its complement object deck (on an alternate input unit) to the load file, if the program is to be loaded. If the LIBE specification is found on this control card, only the $IBLDR card is added to the load file. A $IBLDR card is the first card in the output deck of the Assembler.

Optional Cards

$ID CARD

This control card causes Process Control to transfer to the installation accounting routine (SYSIDR) for accounting purposes.

$IEDIT CARD

Process Control uses the variable field of this control card to set input specifications for the application. It may appear in any group of control cards, and the specifications remain in effect for the remainder of the application unless reset by another $IEDIT card.

$OEDIT CARD

Process Control uses the variable field of this control card to set output specifications for the application. It may appear in any group of control cards, and the specifications remain in effect for the remainder of the application unless reset by another $OEDIT card.

$* CARD

No action besides printing on-line is taken for this control card. It may appear in any group of control cards.

$PAUSE CARD

Processing halts. The Start button must be pressed to proceed.

$ENDREEL CARD

A reel switch is performed between System Input Units SYSIN1 and SYSIN2. 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 Labels levels of IOCS.

Specification Scan

A specification scan may take place if a recognized control card has options that can be specified. Process Control looks for specifications only on $IBJOB, $IBLDR, $IEDIT, and $OEDIT cards. Specifications on $IBFTC, $IBCBC, and $IBMAP cards are scanned by the component program. 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 a specification. The specification is matched against a dictionary, and the proper bit or location is set 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 of the installation is assumed.

IBJOB Processor Maintenance Cards

$DUMP Card

The $DUMP card causes specified portions of system records to be dumped. The format of this control card is as follows:

```
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-space 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 number in octal notation (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.

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 numbers in octal notation. 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. If it is desired to extend the variable field, another $DUMP card that specifies the same system record character and location may be used. The portions of core storage are dumped in the reverse order of their appearance on the card. The $DUMP card specifications are effective only for the application in which they occur.

The first $DUMP card that is read causes the dump routine (JDUMP) to be loaded into core storage starting at $(76237)_8$. 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 Nucleus are reset. The upper core limit in location IBJCOR in the Processor Monitor 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 in core storage when dump requests are made. 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. For example, when the system record IBMAPJ is called into core storage, a dump request may be inserted in IBJOB Processor coding by specifying J11526, starting in column 8 on the $DUMP card.

If system record IBMAPJ were called into core storage more than once after the $DUMP card is 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 IOCS and the Output Editor for processing output. If dump requests are inserted in these areas, a loop occurs.

3. The system record IBLDRQ cannot be dumped, since this record occupies 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 within CALL macros. These restrictions are due to the method used for inserting dump requests. The method is as follows:

   a. The instruction at the location $(xxxxx)_8$ specified on the $DUMP card is saved in a table contained in the dump routine.
   b. A transfer to the dump routine is placed in the specified location.
   c. When the transfer is 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

This control card is used to insert temporary patches in system records, thereby eliminating an unnecessary system edit run. The format of this card is as follows:

```
1          8          16
$PATCH     cxxxxx     instr1, instr2, . . .
```

where the field starting in column 8 contains an alphameric character (c) and a five-digit number in octal notation (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.

The field starting in column 16 contains twelve-digit octal instructions that are to be loaded into core storage starting at location xxxxx. 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.

RESTRICTIONS ON PATCH REQUESTS

Patch requests have the following restrictions:

1. System records IBJOB, JDUMP, IOCSB, IBJOBB, and IBJOBC cannot be patched directly. Another system record must be called, using the sixth character of the system record name. Location xxxxx can then be in the area occupied by one of these records.

2. System record IBLDRQ cannot be patched, since it occupies the same area of core storage as the dump routine, which contains the patch routines.

Error Procedures

When the FORTRAN Compiler, the COBOL Compiler, or the Assembler returns to Process Control, an error word must be left in the accumulator. If no error was detected by the subsystem, the accumulator address and decrement are zero. A suspected machine error is indicated by a nonzero decrement, and a source error is indicated by an error level number in the address. This error level number determines the error procedure used by Process Control, as follows:

LEVEL	PROCEDURE
1	Assemble, if the return is from the FORTRAN or COBOL Compiler, and allow loading if requested.
2	Assemble, if the return is from the FORTRAN or COBOL Compiler, but do not allow loading.
3 or greater	Do not allow assembly or loading.

If no source 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 the positions that they were at 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 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 or not a machine error accompanied the source error.

Error Messages

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 legitimate date. The operator should enter the current date, in BCD, into the keys and press START.

ACTION LABEL INCORRECT.

This message will occur if an argument to action, sent by some part of the system to cause positioning or reading of the system unit, does not match any action table entries.

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.

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, plus an additional word for each set of dump limits on the card.

EOT ON INTERMEDIATE UNIT OR EOB EXIT. ERROR CONDITION.

This message occurs if the system Input/Output Editor, while trying to write 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.

PREST CARD CKSUM ERROR. SEQUENCE NUMBER N.

This message occurs if, while processing a Prest deck, a check sum error is detected. Processing will continue.

PREST CARD SEQ ERROR. SEQUENCE NUMBER N.

This message occurs if, while processing a Prest deck, an error in the sequence of cards is detected. Processing will continue.

PREST CARD FIELD ERROR. SEQUENCE NUMBER N.

This message occurs if, while processing a Prest deck, an error is detected in the field or string count.

XXXXXX 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.

XXXXXX 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 VERSION N HAS CONTROL.

This message occurs each time the IBJOB Processor gains control from the System Monitor.

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.

LINES OUTPUT.

This message occurs following each Processor application. 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.

ONLY SYSIN1, SYSOU1, AND SYSPP1 WILL BE REPOSITIONED FOR RETRY.

This message occurs if machine or system failure is detected by some portion of the IBJOB Processor.

MACHINE OR SYSTEM FAILURE HAS OCCURRED. RETRY IS IMPOSSIBLE. THIS JOB WILL BE CONTINUED.

This message occurs when machine or system failure is detected and if the system input unit is a card reader or if end-of-tape was encountered during the job.

THS JOB WILL BE CONTINUED.

This message occurs if the option to continue is chosen.

REMAINDER OF JOB DELETED.

This message occurs if the option to delete the remainder of the job is chosen.

ASSEMBLY DELETED.

This message occurs if an error in compilation has occurred such that assembly cannot be attempted.

ERROR IN ALTER DECK.

This message occurs when a deck has been altered and an error detected (unused Alter cards, end of file or redundancy while reading Alter cards, or a scan error in an alter card).

BINARY RECORDS (S) ENCOUNTERED WHILE SEARCHING FOR CARDS.

This message occurs when binary records are encountered by the IBJOB Monitor outside of the limits of an object deck.

UNRECOGNIZED OPTION ON ABOVE CARD.

This message occurs when an unrecognized option is encountered on a control card.

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.

ABSMOD ASSEMBLIES CANNOT BE LOADED.

This message occurs when the ABSMOD option is encountered on a $IBFTC, $IBCBC, or $IBMAP card, and loading is requested.

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.

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.

LOADING HAS BEEN SUPPRESSED.
This message occurs when loading has been requested and cannot be performed.

NO PROCESSING THIS P/A.
This message occurs when a processor application contains no deck or $IBLDR card with the LIBE option.

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 SCHF is requested on a $IEDIT card and the system unit function requested for the search is a disk.

PERMANENT REDUNDANCY WHILE READING CONTROL CARDS. THIS P/A CANNOT BE CONTINUED.

****** CARD WITH CORRECT DECK NAME NOT FOUND.
This message occurs when alternate input is requested and the deck requested cannot be found.

PERMANENT REDUNDANCY OR EOT WHILE READING ALTERNATE UNIT.

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.

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).

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).

$IBJOB CARD MISSING.
This message occurs when any control card is encountered before the $IBJOB card for the processor application is encountered.

ALTER FIELD ERROR, COMMA TREATED AS BLANK.
This message occurs when a comma is encountered in columns 1-6 or when a field is written (A*ALTER B. C,). In the latter case, the last comma is treated as a blank.

ALTER FIELD ERROR, CARD AND INSERTIONS IGNORED.
This message occurs when comma or blank is encountered in column 16 or when a field is written (A*ALTER B, ,) or (A*ALTER B,).

EOF OR REDUNDANCY IN ALTER FILE.
This message occurs when 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).

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.

Loader Information

The information contained in this chapter consists of additional reference material pertaining to the Loader. The material can be helpful for a complete understanding of the Loader; nevertheless, this material only supplements the information on the Loader in the section "Programmer's Information."

Organization of the Loader

The Loader is composed of an Initialization section and of five other sections which perform operations necessary to load an object program. These sections are controlled by a load supervisor. The load supervisor is an internal supervisor that is an integral part of the Loader. The initialization section of the Loader receives control from the load supervisor and is responsible for the following:

1. Transferring to the installation accounting routine to record the fact that the Loader is in control.
2. Obtaining units for the Loader to use, defining buffer pools, attaching files to these pools, and opening the files.
3. Scanning the $IBJOB card and storing parameters obtained from it into its communication region.

At the conclusion of the above operations, control is transferred to the load supervisor, which loads Section 1 of the Loader.

The principal task of Section 1 is the processing of control information (control cards, control dictionary, and file dictionary). Section 1 converts this information into a form which is more easily handled by Sections 2 through 5.

Input to Section 1 consists of a file containing a mixture of control information and relative binary text. This file may be on the System Input Unit, or it may be the load file if compiler or Assembler source input is part of the job. The load file, prepared by the Input/Output Editor of the Processor Monitor on the System Utility Unit (SYSUT2) from the deck outputs of compilations or assemblies performed in the job, is merged, by the Processor Monitor, with any binary decks provided as part of the job.

After the processing of Section 1 is completed, control is relinquished to the load supervisor, which initializes the next section of the Loader.

Section 2 of the Loader is concerned with the logical cross-referencing problems of multiple source decks and their required subroutines from the Subroutine Library, and with overlay analysis. It processes the control information tables that are built by Section 1 and builds up the object program file blocks from the $FILE cards stored in that control information storage block.

The principal task of Section 3 is to provide unit assignment for the object program; to give absolute location assignments to each program deck, each subroutine, and the control sections of both; to apportion the unused part of core storage as input/output buffers for the object program; to generate the IOCS calling sequences to define those buffer pools; and to provide a map of the complete object program core storage use. (The map feature of the Loader provides an outline-like picture of the assignment of core storage to the object program).

The input/output unit assignment provides for absolute channel, symbolic channel, and between-application symbolic or reserve channel specification of input/output devices. Provision is also made for absolute assignment of disk areas, drum areas, and Hypertape drives. If necessary, file mounting instructions to the operator are printed by Section 3.

Section 4 of the Loader is read into the core storage area occupied by Section 2. Its main function is to form the final, absolute instructions from the relocatable binary text of the input program and from any subroutine on the library unit which is required by the program.

The input to Section 4 consists of the relocatable binary text of both the input program and subroutines. Input program text may appear as follows:

1. In an internal file.
2. In an internal file and on the System Utility Unit (SYSUT3) — the source text overflow tape, or
3. On the System Utility Units (SYSUT3 and SYSUT4) — the internal text overflow.

Subroutine texts are read from the Subroutine Library tape and are processed in the same manner as program texts. Subroutines are called as determined by their appearance in the required Subroutine Name Table that was formed by Section 2 of the Loader.

The final text is put into an internal file, and onto the System Utility Unit (SYSUT1) if necessary, for pre-execution loading by Section 5 of the Loader. For overlay applications, output can also be on one of the System Library Units. A call to the program to be executed first is generated according to the $ENTRY card or, in its absence, to the section whose name is '......' or, in its absence, to the first program deck encountered.

Section 5, the final phase of the Loader, loads the processed absolute program text into its proper core storage locations and prepares for its execution. The lower half of the program area is set to STRS, and the absolute text contained in the internal file in the upper portion of core storage is scatter-loaded into this lower half. The internal file area is then set to STRS. Absolute text will not be loaded above the locations required by Section 5 for loading the overflow text appearing on the System Utility Unit (SYSUT1). At the completion of the program load, the function of the Loader ends and control is transferred to the generated initialization instructions.

Relocatable Binary Program Deck

A relocatable binary program deck consists of relocatable binary text, the control dictionary, and the file dictionary. This section defines the deck order and format of the relocatable binary text, the control dictionary, and the file dictionary.

Binary Card Format

The following column binary card form is used:

Word 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	0 (Loader or relocatable deck, not Prest)
	5-7	deck type
	8-12	01010
	13-17	word count (beginning with word 3)
	21-35	card sequence number
Word 2	S, 1-35	logical check sum of word 1 and all data words on the card
Words 3-24	S, 1-35	data

Binary Card Sections

A binary program deck is composed of three sections, each prefaced by an alphameric source card identifying the section type. The deck format, exclusive of control cards, is as follows:

COLUMN 1	COLUMN 8
$FDICT	DECKNM
	Binary File Dictionary
$TEXT	DECKNM
	Relocatable Binary Text
$CDICT	DECKNM
	Binary Control Dictionary
$DKEND	DECKNM

Each section of the binary deck (e.g., the control dictionary) and text is sequenced independently, beginning with sequence number 0. Within any section, the cards must be in proper sequence and the sections themselves, if present, must be in order by deck type. The deck type codes, which appear in word 1, bits 5-7, are:

File Dictionary	001
Text	010
Control Dictionary	011

The entire binary portion of the program deck is limited by the $FDICT and $DKEND cards.

Relocatable Binary Text

Words 3, 4, and 5 of the text card are used for up to nineteen 5-bit control groups, one for each subsequent data word on the card. The sign bits of these control words are not used, and the control groups are given in sequence, up to 7 groups per word. Although the card form is column binary, a row binary example is shown for clarity in Figure 16.

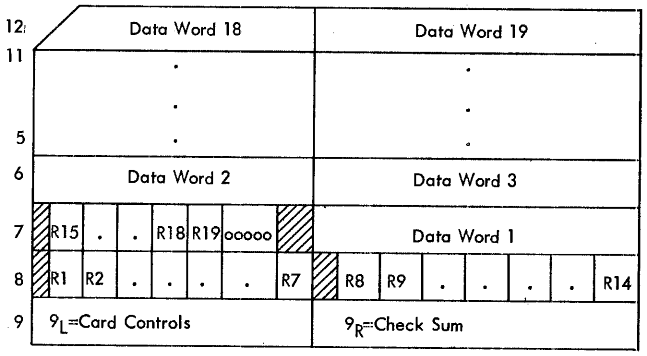


Figure 16. Relocatable Binary Card

The first bit of each control group is used to distinguish between two basic word types:

1	standard data word
0	special entry

STANDARD DATA WORD

The 5-bit control group is of the form 1 AB CD, where:

AB	=	00 constant decrement
	=	01 relocatable decrement
	=	10 the decrement is a dictionary reference (the decrement further defines which type)
	=	11 the decrement is represented by a complex expression

CD has the corresponding code values describing the address. The prefix and tag of a standard data word are constant.

Constant and Simple Relative: The codes 00 and 01 indicate that the field is constant or relative to the program origin, respectively. If the field is constant, no alteration occurs. If the field is relative, a scan is made for its value in the Control Break Table which is formed from the control dictionary of this deck. If that relative location is found, the control dictionary will assign the base to be added; if that relative location is not found, the origin assigned to this deck will be used as the addend. As an example, consider the following instructions, where:

A is relative 100
B is relative 163

SYMBOLIC	RELOCATION BITS	DATA WORD
CLA A	1 00 01	0 50000 0 00100
ADD A+1	1 00 01	0 40000 0 00101
TXH B, 4, A	1 01 01	3 00100 4 00163

Dictionary References: Dictionary references are denoted by a relocation code of 10. The data field of each of these references is a 15-bit code. The high-order bits of the 15-bit code are used to distinguish between types of reference, whereas the low-order part of the field gives the reference number. The reference type codes are:

0000	Control dictionary reference, followed by 11 bits giving the relative location of an entry in the deck control dictionary.
0001	File reference, followed by an 11-bit file index number.
0011, 10, 11	These codes are not assigned.

As an example, consider the following instructions, where the symbol A is the sixth entry in the control dictionary and the file AFILE is the eleventh file of this deck:

SYMBOLIC	RELOCATION BITS	DATA WORD
CLA A, 4	1 00 10	0 50000 4 00006
PZE AFILE	1 00 10	0 00000 0 04013

Complex Fields: The decrement or the address, or both, of a data word can be represented as a complex expression requiring evaluation at load time. A field requiring such evaluation is given a relocation code of 11. The expression to be evaluated may then be expressed in one of two ways:

1. The field equals zero (Long Form Complex). This expression form consists of a string of one or more words, each with a corresponding control group, following the data word. If both the decrement and address are complex, the decrement occurs first.

Each word of a complex expression has the following form:

OP   A, B, C

where:

OP = PZE+
   = PON–
   = PTW*
   = PTH/

The relocation bits within the expression have the same format as that of a standard word except that the 11 code is changed to designate a result storage location. Seven such result storages (0, 1, ... , 6) are used for intermediate results during the evaluation of a complex expression. A word in a complex expression is interpreted as follows:

A OP C $\longrightarrow$ B

where B is the result storage location into which the result is to be placed. The complex expression is terminated by a word for which B=7. As an example, consider the following instruction:

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 dictionary name for TABLE is "TABLE" and that it is the fifth entry in the control dictionary. The instruction would appear as follows:

RELOCATION BITS	DATA WORD	
1 00 11	0 50000 0 00000	
1 00 10	2 00006 1 00005	TABLE*6 $\longrightarrow R_1$
1 00 10	2 00002 2 00005	TABLE*2 $\longrightarrow R_2$
1 11 11	0 00001 1 00002	$R_2+R_1 \longrightarrow R_1$
1 11 00	0 00001 7 00003	$3+R_1 \longrightarrow$ Address of data

2. The field is not equal to zero (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. The 15-bit field is formed as follows:

Bit 1	O, C is added. 1, C is subtracted.
Bits 2-n	relative location of NAME in this deck's control dictionary. As many bits are used as are required to express the total length of the control dictionary; e.g., if CDICT contains 16 entries, then 5 bits would be used, i.e., bits 2-6.
Bits n-15	A (15−n+1)−bit constant to be added to, or subtracted from, the location assigned to the referenced name. Note that the Long Form Complex may have to be used if the length of the control dictionary plus the length of the addend exceeds 14 bits.

As an example, consider the following instruction:

CLA COMMON + 50, 1

where COMMON is a control section whose length is 50.

COMMON is the control dictionary name and is the eighth entry in a 26 entry control dictionary.

RELOCATION BITS	DATA WORD
1 00 11	0 50000 1 10062

The field 10062 is interpreted as follows:

1	2 6	7 15
0	01000	000110010
+	8	50

Special Entry Word

Special entries are used to specify origins, BSS's, VFD's, and other Loader controls. All special entry codes are of the following form:

0 SSSS

where the four bits ssss specify the type of special entry:

0 0000	End of card; no data word is associated with this entry.
0 0001	Location counter control; the data is of the following form: OP A, , relative location (NOTE 1)

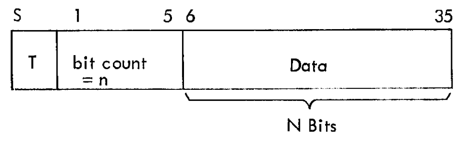
where:

OP = PZE	A is an absolute origin (NOTE 2)
OP = PON	A is relative origin (NOTE 2)
OP = PTW	BSS of length A
OP = PTH	EVEN pseudo operation
OP = MZE	A is a dictionary origin in complex format (NOTE 1)

NOTE 1: Relative location of this instruction as it appears in the listing (OP = MZE).

NOTE 2: Origins are an integral part of text; each card does not carry its relative load address.

0 0010	CALL expansion follows; the data word is of the form PZE 0. This text code and its data word are required for the overlay mechanism of the Loader.
0 0011– 0 0111	are left open for expansion.
0 10VV	VFD expression; the Loader assembles a string of bits or words which contain constants, relative locations, dictionary references, and complex expressions which do not necessarily fall in the normal boundaries of address or decrement. Each data word which corresponds to a VFD control group specifies one field of the VFD expression and specifies whether a word should be terminated. The data word general format is:



T = 0	The assembly continues in the same machine word or words (across machine word boundaries).
T = 1	The current machine word is terminated and filled with zeros in the unused right positions.
N ≦ 30	Specifies that the rightmost n bits of this word are to be inserted into the generated string (see VV codes 00 – 11).
0 10 00	VFD, n bits are absolute and are to be inserted into the string.
0 10 01	VFD, the address of a data word, is a relative location. It is relocated as any other relative (01) field and is substituted for a 15-bit address. Rightmost n bits are then inserted into the VFD string.
0 10 10	VFD, the address of a data word, is a dictionary reference. It is evaluated as any dictionary reference (10) field. The action is the same as with VFD type 01.
0 10 11	VFD, the address of a data word, conforms to the rules for complex expressions; after evaluation, the substitutional action is the same as with VFD type 01.

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

assume relative location A = 103
assume dictionary location B = 3
assume dictionary size = 25

0 1100 0 1110	are left open for possible expansion.
0 1111	end of text; the address of the corresponding data word contains the relative location of the first instruction to be executed if this deck is named on a $ENTRY card.

Control Dictionary

The control dictionary defines, in a deck, procedure and/or data areas which may be deleted, replaced, or referred to by other program segments which have been separately assembled or compiled. Each entry in the control dictionary supplies an external reference name, its relative location to the origin of the program, and its length. The control dictionary format is:

word 1 =	BCI	1 , EXNAME
word 2 =	PFX	L, , N
PFX =	PZE	The section is a real section (it exists in this deck).
=	PON	The section is an EVEN pseudo-operation; the length always equals 0; EXNAME is zeros.
=	PTW	The section is a virtual section (only references to the section appear); hence, the section must be defined at load time.
L =		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 cards except the last must be full.

Control dictionary entries are ordered by increasing relative location and, within this, by decreasing length.

FORMAT PREFACE ENTRY

The first entry in the control dictionary is a special entry giving the location of the first executable instruction, the program length, the machine to be assembled on, and the size of the control dictionary (power of 2 is given). The contents of the two words are:

```
PFX  X, , N
PZE  P, , MACH
```

X	=	The relative location of the first executable instruction.
N	=	The program length.
P	=	The power of 2, which includes the number of entries in this control dictionary.
MACH	=	0, assembled for 7090.
	=	4, assembled for 7094 (may not be run on 7090).
PFX = PZE		Normal relocatable deck.
= MON		Deck contains no relative origin. X is taken as the absolute location of this deck.

File Dictionary

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 those items which are unchanged by the generated object program; hence, they are recorded in the file dictionary to ensure that the programmer does not change these through modification of $FILE cards.

CARD FORMAT

Five words of text are required for each file referred to in a program.

Word 1	standard 9L format, deck type 001
Word 2	check sum
Words 3-7	file check entry i
Words 8-12	file check entry i+1
	etc.
	etc.
	etc.
Words 23-24	not used

One card contains 20 data words located in words 3 through 22.

File Dictionary Entry Specifications: The format of the five words is:

WORDS	BITS	CONTENTS
1	S	If = 1, mixed mode file
	1	If = 1, last FDICT entry
	2-5	0
	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	0

	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
		If = 001, n ≤ 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 non-standard label routine
3-5		18-character file name

Library Search

A Subroutine Library search procedure is initiated by the Loader if any virtual control section has not been defined after all object program control dictionaries have been processed.

The Subroutine Library consists of 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 subroutine's position in the control information and text files.

List 2: The dependence list contains, for each entry in the control section name list, 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 (i.e., 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 by specifying, in the object program, 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. Each dependency list is punctuated by a 3-bit operation code (prefix and/or tag).

OCTAL		
0	[	Beginning of dependency list
4	]	End of list
2	,	Separator

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 library subroutine 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 acquisition of needed subroutines without multipass searching of the library unit.

Even Storage Feature

Because of the 7094 machine requirements for the storage of double-precision floating-point operands in successive locations (the first part of the number having an even machine location) and because of the increased efficiency which may be gained by placing certain instruction sequences in even or odd locations, the Loader provides a technique to ensure that an even storage location is assigned to specified data or instructions. The specification of even storage for data or instructions is accomplished by the use of the IBMAP EVEN pseudo-operation.

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. Except in an absolute assembly, the assembly process should not generate any data to force an even relative location counter.

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.

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 the execution of the program. Since the AXT instruction, if generated, always occupies odd storage, its execution is normally free.

There are certain instances in which the programmer may encounter difficulty when using the EVEN storage pseudo-operation. The most frequent usages are: indexing through an array containing an EVEN pseudo-operation; and not computing the length of a block of data containing EVEN pseudo-operations by subtracting the symbolic location of the first item from the location of the last. Since the Loader may generate AXT instructions, the length of blocks of data and/or instructions may change, being dependent on the origin assigned to the program and on the actions taken upon the control sections preceding the EVEN pseudo-operations.

Format EVEN Control Dictionary Entry

```
word 1     BCI     1 , 000000
     2     PON     R, , 0
```

R is the relative location which must be assigned an even absolute location.

Format EVEN Text Entry

```
    RELOCATION BITS          DATA WORD
       0 0001                PTH   A
```

A conforms with the 15-bit code for control dictionary references (such as 10) and refers to the correct EVEN control section.

EVEN Program Example

```
REL LOC RELOCATION
  CTR      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     O
  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 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
```

Error Messages

ABS.PROG.LD.FAILS.NO.EXEC.

Permanent redundancy occurred in loading final text overflow tape.

A SUBROUTINE TO BE INSERTED HAS THE SAME NAME AS AN EXISTING SUBROUTINE WHICH HAS NOT BEEN DELETED.

CALL TO OBJECT PROGRAM UNDEFINED.

$***** CARD ENCOUNTERED, RETURNING TO MONITOR FOR PROCESSING OF THIS CARD.

$JOB,$IBSYS,$EXECUTE,$STOP card has been read.

CONTROL DICTIONARY CONTAINS UNDEFINED VIRTUAL.

Printed only if LOGIC is specified and the Control Dictionary contains an undefined virtual.

CONTROL SECTION '******' IS AN UNDEFINED ENTRY POINT.

CONTROL SECTION '******' REQUIRED BY SUBROUTINE '******' IS VIRTUAL IN THE SUBROUTINE LIBRARY.

CONTROL SECTION '******' SPECIFIED ON $USE CARD WAS DELETED. TEXT ERRORS MAY OCCUR.

CYLINDER COUNT SPECIFIED EXCEEDS 250.

A maximum of 250 cylinders is permitted on a $FILE card.

DECK '******' DOES NOT EXIST IN SOURCE INPUT. $OMIT ENTRY IS IGNORED.

A specification on a $OMIT card is in error.

DECK '******' DOES NOT EXIST IN SOURCE INPUT. SECTION RENAME IS IGNORED.

A specification on a $NAME card is in error.

DECK '******' DOES NOT EXIST IN SOURCE INPUT. $USE ENTRY IS IGNORED.

A specification on a $USE card is in error.

DECK FORMAT ERROR – PROCESSING DECK '******' A CARD SEQUENCE BREAK IN '******'. SEQUENCE NUMBER ******

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 ******

DECK FORMAT ERROR – PROCESSING DECK '******' ***** BINARY CARD IS NOT IN PROPER PLACE. CANNOT FOLLOW ****** CARD.

DECK FORMAT ERROR – PROCESSING DECK '******'. ***** BINARY CARD IS NOT IN PROPER PLACE. THE LAST CORRECT CARD IS ***** – SEQUENCE NUMBER ******

DECK FORMAT — PROCESSING DECK '******' CARD IS NOT IN PROPER PLACE. THIS CARD IGNORED.

DECK FORMAT ERROR – PROCESSING DECK '******' CARD IS NOT IN PROPER PLACE. THE LAST CORRECT CARD IS ***** – SEQUENCE NUMBER ******

DECK FORMAT ERROR – PROCESSING DECK '******' CHECKSUM ERROR – DOES NOT COMPARE IN BITS ************ THE LAST CORRECT CARD IS ***** – SEQUENCE NUMBER ******

DECK FORMAT ERROR – PROCESSING DECK '******' ************ IS AN ILLEGAL 9L FORMAT THE LAST CORRECT CARD IS ***** – SEQUENCE NUMBER ******

DECK FORMAT ERROR – PROCESSING DECK '******' TEXT ENCOUNTERED IN READING CONTROL INFORMATION.

Subroutine library format error-text in middle of control information part of a subroutine.

DECK ****** IS ASSEMBLED FOR IBM 7094 AND CANNOT BE RUN ON IBM 7090.

DECK NAME APPEARING ON THE ABOVE CARD DOES NOT AGREE WITH NAME FROM $IBLDR CARD.

DECKNAME CONTAINS ILLEGAL CHARACTER OR BLANK. A DECKNAME OF ALL BLANKS WILL BE USED.
    Deck name on $IBLDR card is missing or in error.

DECK FORMAT ERROR – PROCESSING DECK (******)

DECK NAME '******' ON $GROUP CARD IS IGNORED.

DECK NAME '******' ON $POOL CARD IS IGNORED.

DECK NAME ON $TEXT OR $DKEND CARD UNRECOGNIZABLE.
    No $IBLDR card appeared with the above name.

DISREGARD MOUNTING INSTRUCTIONS.
    Printed on-line when nogo is set after file list has been printed on-line.

END OF FILE NOT PERMITTED AT THIS POINT.

END OF TAPE CONDITION OCCURRED IN WRITING 'SUBROUTINE LIBRARY'.
    Retry subroutine edit.

ENTRY POINT SPECIFIED IS NOT IN MAIN LINK.

ERROR IN COMPLEX OPERATOR AT REL LOC XXXXX, TEXT FOLLOWING MAY BE IN ERROR. (DECK 'DECKNM').

ERROR IN COMPLEX RESULT STORAGE REF AT REL LOC XXXXX, TEXT FOLLOWING MAY BE IN ERROR. (DECK 'DECKNM').

ERROR IN FILE NAME ENCOUNTERED ON A $LABEL CARD. THE CARD WILL BE IGNORED.

ERROR IN VARIABLE FIELD OF ABOVE CARD. THE FIELD IS IGNORED.

$ETC CARD NOT FOLLOWING $FILE CARD WHEN REQUIRED. ERRORS MAY OCCUR.

$FILE CARD ACTIVITY SPECIFIED EXCEEDS 99. THIS MAXIMUM WILL BE USED.

$FILE CARD BLOCK SIZE SPECIFIED EXCEEDS 9999. THIS MAXIMUM WILL BE USED.

FILE '******************' BASE OF 'BLOCK SIZE A MULTIPLE OF' IS INCONSISTENTLY SPECIFIED.
    File Dictionaries from different decks (for same file) do not agree.

FILE ****************** CHANNEL IS ILLEGITIMATE (EXECUTION IS NOT ALLOWED)

FILE CHECK – FILE '******************' DEVIATION FROM BASE OF 'BLOCK SIZE A MULTIPLE OF'.
    File Dictionary does not agree with $FILE card for the same file.

FILE CHECK – FILE '******************' DEVIATION FROM 'EXACT BLOCK SIZE'.
    File Dictionary does not agree with $FILE card for the same file.

FILE CHECK – FILE '******************' DEVIATION FROM FILE TYPE.
    File Dictionary does not agree with $FILE card for the same file.

FILE CHECK – FILE '******************' DEVIATION FROM 'MINIMUM BLOCK SIZE'.
    File Dictionary does not agree with $FILE card for the same file.

FILE CHECK – FILE '******************' DEVIATION FROM MIXED MODE.
    File Dictionary does not agree with $FILE card for the same file.

FILE CHECK – FILE '******************' DEVIATION FROM MODE.
    File Dictionary does not agree with $FILE card for the same file.

FILE CHECK – FILE '******************' DEVIATION FROM UNIT ASSIGNMENT (CARD UNIT NOT ALLOWED).

FILE CHECK – FILE '******************' DEVIATION FROM UNIT ASSIGNMENT (TAPE UNIT NOT ALLOWED).

FILE '******************' 'EXACT BLOCKSIZE' IS INCONSISTENTLY SPECIFIED.
    File Dictionaries from different decks (for same file) do not agree.

FILE ****************** ILLEGAL SECONDARY UNIT (REQUEST IS IGNORED), SECOND UNIT ASSIGNED SAME AS FIRST.

FILE ****************** ILLEGAL SYSUNI CODE-REPORT

FILE ****************** INTERSYSTEM INPUT FILE HAS NOT BEEN RESERVED (EXECUTION IS NOT ALLOWED)

FILE '******************' I/O UNIT TYPE REQUIREMENT IS INCONSISTENTLY SPECIFIED.
    File Dictionaries from different decks (for same file) do not agree.

FILE '******************' MODE OR FILE I/O TYPE IS INCONSISTENTLY SPECIFIED
    File Dictionaries from different decks (for same file) do not agree.

FILE ****************** PRINTER ILLEGAL AS INPUT (EXECUTION IS NOT ALLOWED)

FILE ****************** PROCESSING ERROR (TIG)
    Machine or system error during unit assignment.

FILE ****************** PUNCH ILLEGAL AS INPUT (EXECUTION IS NOT ALLOWED)

FILE ****************** READER ILLEGAL AS OUTPUT (EXECUTION IS NOT ALLOWED)

FILE RENAME FOR FILE '******************' IS IGNORED. DECK '******' DOES NOT EXIST.

FILE RENAME FOR FILE '******************' IS IGNORED. FILE DOES NOT EXIST IN ANY FILE DICTIONARIES.

FILE RENAME FOR FILE '******************' IS IGNORED. FILE DOES NOT EXIST IN DECK '******'

FILE ****************** RESERVE UNIT NAME IS ILLEGAL (EXECUTION IS NOT ALLOWED)

FILE '******************' SPECIFIED ON $GROUP CARD DOES NOT EXIST.

FILE '******************' SPECIFIED ON $LABEL CARD DOES NOT EXIST.

FILE '******************' SPECIFIED ON $POOL CARD DOES NOT EXIST.

FILE ****************** UNIT ****** ILLEGAL AS INPUT (EXECUTION IS NOT ALLOWED)

FILE ****************** UNIT ****** ILLEGAL AS OUTPUT (EXECUTION IS NOT ALLOWED)

FILE ****************** UNIT ****** ILLEGAL FOR BCD MODE USE (STANDARD OPTION IS ASSUMED)

File mode is set binary.

FILE ****************** UNIT ****** ILLEGAL FOR BINARY MODE USE (STANDARD OPTION IS ASSUMED)

File mode is set BCD.

FILE ****************** UNIT ****** IS AN ILLEGAL SECONDARY UNIT (REQUEST IS IGNORED), SECOND UNIT ASSIGNED SAME AS FIRST.

FILE ****************** UNIT ****** NOT ALLOWED FOR LABELLED FILE USE. (EXECUTION IS NOT ALLOWED)

FILE ****************** UNIT REQUESTED IS NOT AVAILABLE (EXECUTION IS NOT ALLOWED)

FILE ****************** UNIT2 CHANNEL IS ILLEGITIMATE (REQUEST IS IGNORED)

FILE ****************** UNIT2 REQUESTED IS NOT AVAILABLE (REQUEST IS IGNORED)

FIRST CARD READ FROM 'INPUT'/'GO TAPE' IS NOT A $IBJOB CARD.

FORMAT ERROR ENCOUNTERED ON A $LABEL CARD. THE CARD WILL BE IGNORED.

FORMAT ERROR ENCOUNTERED ON A $SIZE CARD. THE CARD WILL BE IGNORED.

FORMAT ERROR FOR FIELD '******' OF $NAME CARD. THE REMAINDER OF THIS CARD AND ASSOCIATED $ETC CARDS WHICH FOLLOW WILL BE IGNORED.

FORMAT ERROR FOR FIELD '******' OF $OMIT CARD. THE REMAINDER OF THIS CARD AND ASSOCIATED $ETC CARDS WHICH FOLLOW WILL BE IGNORED.

FORMAT ERROR FOR FIELD '******' OF $USE CARD. THE REMAINDER OF THIS CARD AND ASSOCIATED $ETC CARDS WHICH FOLLOW WILL BE IGNORED.

$GROUP CARD BUFFER COUNT SPECIFIED EXCEEDS 999. THE FIELD WILL BE OMITTED.

$GROUP CARD OPEN FILE COUNT SPECIFIED EXCEEDS 99. THE FIELD WILL BE OMITTED.

$IBLDR CARD ENCOUNTERED WHICH SPECIFIES 'LIBE' DURING PROCESSING OF 'NOLIBE' OPTION CARDS ONLY. THE CARD WILL BE IGNORED.

$IBLDR CARD ENCOUNTERED WHICH SPECIFIES 'NOLIBE' DURING PROCESSING OF 'LIBE' OPTION CARDS ONLY. THE CARD WILL BE IGNORED.

$IBLDR CARD ENCOUNTERED WHILE PROCESSING SUBROUTINE WHICH HAS THE SAME NAME AS $IBLDR CARD FROM SOURCE INPUT WHERE 'LIBE' OPTION WAS NOT SPECIFIED.

$IBLDR CARD WITH DUPLICATE NAME ENCOUNTERED WHILE PROCESSING SOURCE INPUT.

ILLEGAL BCD VALUE ENCOUNTERED ON A $ETC CARD FOLLOWING A $FILE CARD. THE $FILE CARD AND ASSOCIATED $ETC CARDS WILL BE IGNORED.

ILLEGAL BCD VALUE ENCOUNTERED ON A $ETC CARD FOLLOWING A $GROUP CARD. THE $GROUP CARD AND ASSOCIATED $ETC CARDS WILL BE IGNORED.

ILLEGAL BCD VALUE ENCOUNTERED ON A $ETC CARD FOLLOWING A $POOL CARD. THE $POOL CARD AND ASSOCIATED $ETC CARDS WILL BE IGNORED.

ILLEGAL BCD VALUE ENCOUNTERED ON A $FILE CARD. THIS CARD AND ASSOCIATED $ETC CARDS WILL BE IGNORED.

ILLEGAL BCD VALUE ENCOUNTERED ON A $GROUP CARD. THIS CARD AND ASSOCIATED $ETC WILL BE IGNORED.

ILLEGAL BCD VALUE ENCOUNTERED ON A $POOL CARD. THIS CARD AND ASSOCIATED $ETC CARDS WILL BE IGNORED.

ILLEGAL CHARACTER ENCOUNTERED ON A $ETC CARD FOLLOWING A $FILE CARD. THE $FILE CARD AND ASSOCIATED $ETC CARDS WILL BE IGNORED.

ILLEGAL CHARACTER ENCOUNTERED ON A $ETC CARD FOLLOWING A $GROUP CARD. THE $GROUP CARD AND ASSOCIATED $ETC CARDS WILL BE IGNORED.

ILLEGAL CHARACTER ENCOUNTERED ON A $ETC CARD FOLLOWING A $POOL CARD. THE $POOL CARD AND ASSOCIATED $ETC CARDS WILL BE IGNORED.

ILLEGAL CHARACTER ENCOUNTERED ON A $FILE CARD. THIS CARD AND ASSOCIATED $ETC WILL BE IGNORED.

ILLEGAL CHARACTER ENCOUNTERED ON A $GROUP CARD. THIS CARD AND ASSOCIATED $ETC CARDS WILL BE IGNORED.

ILLEGAL CHARACTER ENCOUNTERED ON A $POOL CARD. THIS CARD AND ASSOCIATED $ETC CARDS WILL BE IGNORED.

ILLEGAL CHARACTER. REMAINDER OF CARD IGNORED

$ORIGIN or $INCLUDE card contains illegal character or a blank in column 16.

ILLEGAL FILE NAME ENCOUNTERED ON A $ETC FOLLOWING A $FILE CARD. THE $FILE CARD AND ASSOCIATED $ETC CARDS WILL BE IGNORED.

ILLEGAL FILE NAME ENCOUNTERED ON A $ETC CARD FOLLOWING A $GROUP CARD. THE $GROUP CARD AND ASSOCIATED $ETC CARDS WILL BE IGNORED.

ILLEGAL FILE NAME ENCOUNTERED ON A $ETC CARD FOLLOWING A $POOL CARD. THE $POOL CARD AND ASSOCIATED $ETC CARDS WILL BE IGNORED.

INPUT/OUTPUT ERROR – BLOCK SEQUENCE FAILURE AND PERMANENT REDUNDANCY OCCURRED IN READING LIBRARY TEXT FILE

Machine or system error.

INPUT/OUTPUT ERROR – UNEXPECTED END OF FILE IN READING INTERMEDIATE TEXT.

INPUT/OUTPUT ERROR – UNEXPECTED END OF FILE IN READING LIBRARY CTRL FILE

INPUT/OUTPUT ERROR – UNEXPECTED END OF FILE IN READING LIBRARY TEXT FILE

INSUFFICIENT STORAGE FOR CONTROL DICTIONARIES AND CONTROL INFORMATION DETECTED BY ******.

If not a machine error, this is the most serious error possible.

PROGRAM IS TOO LARGE FOR THE LOADER TO HANDLE.

INSUFFICIENT STORAGE TO GENERATE SUBROUTINE SECTION NAME TABLE AND SUBROUTINE DEPENDANCE TABLE.

I/O ERROR – EOB IN WRITING FINAL TEXT.

Machine or system error.

I/O ERROR – TEXT – EOB.
Machine or system error.

I/O ERROR – TEXT – PERM. REDUNDANCY.
Machine or system 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 1000 unique control section names are allowed.

LOADING TERMINATED DUE TO IMPROPERLY DEFINED OVERLAY STRUCTURE.

ILLEGAL FILE NAME ENCOUNTERED ON A $FILE CARD. THIS CARD AND ASSOCIATED $ETC CARDS WILL BE IGNORED.

ILLEGAL FILE NAME ENCOUNTERED ON A $GROUP CARD. THIS CARD AND ASSOCIATED $ETC WILL BE IGNORED.

ILLEGAL FILE NAME ENCOUNTERED ON A $POOL CARD. THIS CARD AND ASSOCIATED $ETC CARDS WILL BE IGNORED.

ILLEGAL SECTION NAME ENCOUNTERED ON A $ENTRY CARD. THE CARD WILL BE IGNORED.

IMPROPER FORMAT
Leading, trailing, or multiple field separators in the variable field of $ORIGIN or $INCLUDE card.

IMPROPER SYMBOLIC ORIGIN
The symbolic origin on a $ORIGIN card is all numeric or greater than six characters.

INPUT/OUTPUT ERROR – BLOCK SEQUENCE FAILURE AND PERMANENT REDUNDANCY OCCURRED IN READING GENERATED CIF
Machine or system error.

INPUT/OUTPUT ERROR – BLOCK SEQUENCE FAILURE AND PERMANENT REDUNDANCY OCCURRED IN READING GENERATED TIF
Machine or system error.

INPUT/OUTPUT ERROR – BLOCK SEQUENCE FAILURE AND PERMANENT REDUNDANCY OCCURRED IN READING INTERMEDIATE TEXT
Machine or system error.

INPUT/OUTPUT ERROR – BLOCK SEQUENCE FAILURE AND PERMANENT REDUNDANCY OCCURRED IN READING LIBRARY CTRL FILE
Machine or system error.

INPUT/OUTPUT ERROR – BLOCK SEQUENCE FAILURE AND PERMANENT REDUNDANCY OCCURRED IN READING LIBRARY SRNT/SRDT
Machine or system error.

LOADING TERMINATED DUE TO TOO MANY VIRTUAL SECTIONS.
If not a system or machine error, must be reassembled. Current limit is 350.

NO DECKNAME IN SPECIFICATION ON $USE CARD. '******' ENTRY IS IGNORED.

NONSTANDARD LABEL ROUTINE FOR FILE '******************' WAS DELETED BY LOAD CONTROL CARDS.

NO SYMBOLIC ORIGIN SPECIFIED.
Symbolic origin is not first in variable field of $ORIGIN card.

INPUT/OUTPUT ERROR – BLOCK SEQUENCE FAILURE IN READING GENERATED CIF
Machine or system error.

INPUT/OUTPUT ERROR – BLOCK SEQUENCE FAILURE IN READING GENERATED TIF
Machine or system error.

INPUT/OUTPUT ERROR – BLOCK SEQUENCE FAILURE IN READING INTERMEDIATE TEXT.
Machine or system error.

INPUT/OUTPUT ERROR – BLOCK SEQUENCE FAILURE IN READING LIBRARY CTRL FILE
Machine or system error.

INPUT/OUTPUT ERROR – BLOCK SEQUENCE FAILURE IN READING LIBRARY SRNT/SRDT
Machine or system error.

INPUT/OUTPUT ERROR – BLOCK SEQUENCE FAILURE IN READING LIBRARY TEXT FILE.
Machine or system error.

INPUT/OUTPUT ERROR – END OF BUFFERS CONDITION OCCURRED IN READING GENERATED CIF
Machine or system error.

INPUT/OUTPUT ERROR – END OF BUFFERS CONDITION OCCURRED IN READING GENERATED TIF
Machine or system error.

INPUT/OUTPUT ERROR – END OF BUFFERS CONDITION OCCURRED IN READING 'INPUT'/'GO TAPE'
Machine or system error.

INPUT/OUTPUT ERROR – PERMANENT REDUNDANCY OCCURRED IN READING GENERATED CIF
Machine or system error.

INPUT/OUTPUT ERROR – PERMANENT REDUNDANCY OCCURRED IN READING GENERATED TIF.
Machine or system error.

INPUT/OUTPUT ERROR – PERMANENT REDUNDANCY OCCURRED IN READING ('INPUT'/'GO TAPE'
Machine or system error.

INPUT/OUTPUT ERROR – PERMANENT REDUNDANCY OCCURRED IN READING INTERMEDIATE TEXT.
Machine or system error.

INPUT/OUTPUT ERROR – PERMANENT REDUNDANCY OCCURRED IN READING LIBRARY CTRL FILE.
Machine or system error.

INPUT/OUTPUT ERROR – PERMANENT REDUNDANCY OCCURRED IN READING LIBRARY SRNT/SRDT
Machine or system error.

INPUT/OUTPUT ERROR – PERMANENT REDUNDANCY OCCURRED IN READING LIBRARY TEXT FILE
Machine or system error.

INPUT/OUTPUT ERROR – UNEXPECTED END OF FILE IN READING (INPUT'/'GO TAPE'). NOT ENOUGH UNITS AVAILABLE
General error message if not enough units to complete total unit assignment requests.

'NOTEXT' IGNORED BECAUSE 'LIBE' OPTION IS SPECIFIED.

NOT IBLOADER CONTROL CARD. CARD IGNORED.

OBJECT PROGRAM EXCEEDS AVAILABLE STORAGE.

OPTIONS OTHER THAN ABSOLUTE ORIGIN IGNORED ON MAIN LINK $ORIGIN CARD

ORIGIN IS INCORRECTLY SPECIFIED. ORIGIN IS IGNORED.

ORIGIN MUST BE SPECIFIED ON $ASSIGN CARD.

OVERLAY SUBROUTINE LOVRY NOT DEFINED.

PARAMETER '******' NO RECOGNIZED. IGNORED
  Unrecognizable parameter found on $ORIGIN card.

$POOL CARD BLOCK SIZE SPECIFIED EXCEEDS 9999. THE FIELD WILL BE OMITTED.

$POOL CARD BUFFER COUNT SPECIFIED EXCEEDS 999. THE FIELD WILL BE OMITTED.

POOLING ERROR GROUPING FILE '******************' '******' PREVIOUSLY SPECIFIED, IGNORED
  Section or deck appears more than once on $INCLUDE cards for the same link.

PROCESSING ERROR – LNSC FOR CONTROL SECTION '******' CONTAINS INVALID ADDRESS.
  Machine or system error.

PROGRAM EXCEEDS ABSOLUTE LOCATION ******.
  Program may use but not load above specified value.

RELATIVE LOCATION OF TEXT CONTAINING ILLEGAL CONTROL GROUP. (DECK '******').
  A list of relative locations follows.

RELATIVE LOCATION OF TEXT CONTAINING ILLEGAL LOCATION COUNTER CONTROL. (DECK '******').

RELATIVE LOCATION OF TEXT CONTAINING UNDEFINABLE FIELD. (DECK '******').
  A list of relative locations follows.

******, REQUIRED AS A LOAD-TIME DEPENDENCY BY ROUTINE, IS VIRTUAL IN IBLIB.

LIBRARIAN PROCESSING CONTINUES WITH THIS DEPENDENCY IGNORED.

SECONDARY $ENTRY CARD ENCOUNTERED AND IGNORED.

SECTION ****** IS AN UNDEFINED SYSTEM SYMBOL.
  Faulty system subroutines or machine error.

SECTION '******' DOES NOT EXIST IN DECK '******'. $OMIT ENTRY IS IGNORED.
  A specification on a $OMIT card is in error.

SECTION '******' DOES NOT EXIST IN DECK '******'. SECTION RENAME IS IGNORED.
  A specification on a $NAME card is in error.

SECTION '******' DOES NOT EXIST IN DECK ('******'. $USE ENTRY IS IGNORED.

SECTION '******' DOES NOT EXIST IN SOURCE INPUT. $OMIT ENTRY IS IGNORED.
  A specification on a $OMIT card is in error.

SECTION '******' DOES NOT EXIST IN SOURCE INPUT. SECTION RENAME IS IGNORED.
  A specification on a $NAME card is in error.

SECTION '******' IN DECK '******' HAS BEEN MARKED FOR DELETION AND CANNOT BE SPECIFIED ON $USE CARD.
  A specification on a $USE card is in error.

SECTION – 2 I/O ERROR EOB IN SRDICT.

SECTION – 2 I/O ERROR EOF IN SRDICT.

SECTION NAME OF '000000' OR '//' CANNOT BE SPECIFIED. $NAME ENTRY IS IGNORED.

SECTION NAME OF '000000' OR '//' CANNOT BE SPECIFIED. $OMIT ENTRY IS IGNORED.

SECTION NAME OF '000000' OR '//' CANNOT BE SPECIFIED. $USE ENTRY IS IGNORED.

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 STARTING CYLINDER POSITION IS GREATER THAN 249. ZERO WILL BE USED.

STORAGE ALLOCATION ERROR – BUFFER COUNT SPECIFIED ON A POOL CARD IS INSUFFICIENT.

STORAGE ALLOCATION ERROR – INSUFFICIENT INPUT/OUTPUT BUFFER STORAGE.

SUBROUTINE DICTIONARY FORMAT ERROR.
  1. End of file in middle of subroutine section name table.
  2. No subroutine section name table.
  3. Subroutine section name table not complete.
  4. Subroutine section name table has invalid format.
     a. Invalid operation.
     b. Comma encountered when bracket count is not greater than zero.
     c. Too many right brackets.

SUBROUTINE NAME IS INCORRECTLY SPECIFIED.
  Invalid format of deck name on Librarian Control Card.

SYSTEM ERROR OR CPU MAINFRAME FAILURE. ENTRY IN SUBROUTINE SECTION NAME TABLE DOES NOT APPEAR AS REAL RETAINED SECTION IN ANY CONTROL DICTIONARY.

THE ABOVE CARD IS NOT A LIBRARIAN CONTROL CARD.

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.
  Machine or system error.

UNDEFINED CONTROL DICTIONARY ENTRIES REFERENCED.
  A list of control section names always follows.

UNDEFINED FILE '******************' $FILE CARD IS MISSING.

UNDEFINED SECTION OR DECK NAME '******'.
  Section or deck name on $INCLUDE card is undefined.

UNDEFINED VIRTUAL CONTROL SECTION '******'
  Not printed if logic requested.

UNEXPECTED END OF BUFFERS CONDITION ENCOUNTERED IN WRITING OF INTERMEDIATE TEXT FILE.

UNIT SYSXXX NOT ATTACHED AND READY
  Unit specified for overlay links has not been attached to a physical drive.

UNRECOGNIZABLE PARAMETER ENCOUNTERED ON A $FILE CARD. '******' WILL BE IGNORED.

UNRECOGNIZABLE PARAMETER ENCOUNTERED ON A $GROUP CARD. '******' WILL BE IGNORED.

UNRECOGNIZABLE PARAMETER ENCOUNTERED ON A $IBLDR CARD. '******' WILL BE IGNORED.

UNRECOGNIZABLE PARAMETER ENCOUNTERED ON A $LABEL CARD. THE FIELD IS STORED AS ZERO.

UNRECOGNIZABLE PARAMETER ENCOUNTERED ON A $POOL CARD. '******' WILL BE IGNORED.

UNRECOGNIZABLE PARAMETER ON $IBJOB CARD. ****** IS IGNORED.

VALUE SPECIFIED ON $SIZE CARD EXCEEDS FIELD SIZE. THE CARD WILL BE IGNORED.

VIRTUAL SECTION NAME LIST IS FULL.
If not system or machine error, the loader must be reassembled. Current limit is 350.

Assembler Information

A Brief Discussion of the Assembly Process

The information in this section is provided to give the reader an understanding of the basic structure of the Assembler. The core storage layout diagram, shown in Figure 17, should be used in conjunction with the following paragraphs.

The Assembler is divided into two major phases: Phase 1 performs a pass over the source input to form the dictionary; Phase 2 performs a pass to form the assembled instructions.

The time between passes is used to determine the values assigned to the various symbols. The second pass is made over an internal form of binary information, rather than over the BCD information of the original source.

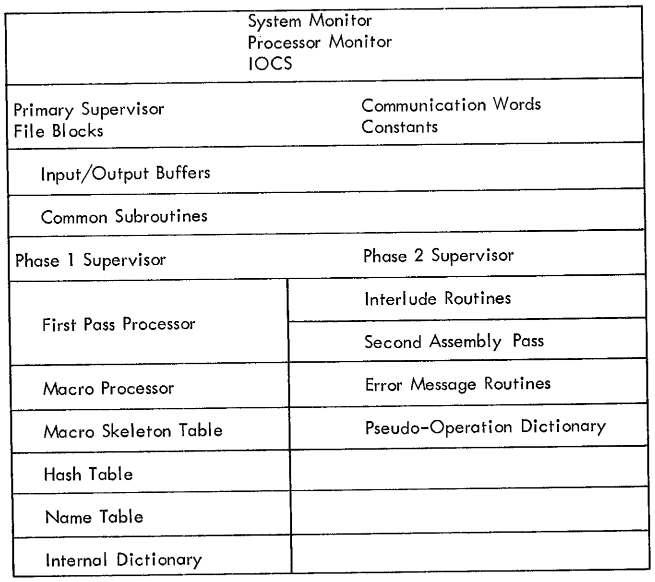


Figure 17. Core Storage Layout

Initialization

The initialization routine receives control directly from the IBJOB Monitor. It performs the following three functions:

1. The System Unit Assignment Table in the System Monitor is examined to find the physical units to be used by the IBJOB Processor.

2. The $IBMAP Card is scanned, and modal switches are set according to the information in the variable field.

3. The operation codes are placed into the Main Dictionary. Operations are not included into their final location in the Main Dictionary because:
   a. Alteration of the algorithm used for symbol storage and retrieval alters the dictionary location of the operations.
   b. Additions, deletions, and insertions of individual operations in the assembled operation table do not affect the entire table; nor does the table need to be in any particular order.

Upon completion of these operations, control is passed to the Primary Supervisor, which, in turn, passes control to the Phase 1 Supervisor. The initialization routines are loaded with Phase 1, but are subsequently overlaid by the Main Dictionary.

Phase 1

In this phase, a pass over the source deck is made and the following functions are performed:

1. The Main Dictionary is constructed. Items entered into this dictionary are: symbols in the location field, qualification symbols, location counter symbols, and any new operations which may be defined.

The Main Dictionary, which is the Assembler's main information table, consists of the following three parts:
   a. The Name Table, n words in length, which contains the BCD representation for every symbol in the program referred to by internal text.
   b. The Internal Dictionary, n words in length, which contains explicit information about the symbol that is necessary for processing.
   c. The Hash Table, n/2 words in length, which provides the necessary link between the scattered information of the Name Table and the linear information of the Internal Dictionary.

2. The pseudo-operation dictionary is constructed. Items in this dictionary consist essentially of the variable fields of any pseudo-operation that may affect a location counter.

3. Literals are converted to binary and stored in the literal pool.

4. Macro-definitions are placed in the Macro Skeleton Table. Card images required by the expansion of a macro-instruction are produced.

5. Card images required by a DUP sequence are produced.

6. The truth value of an IFF or IFT statement is evaluated to determine whether or not to scan the following card.

7. The internal text corresponding to each card is produced. The original card images are retained only for listing purposes. They are deleted in any situation for which listing is not required.

Processing in this phase is parallel, i.e., more than one of the previously described functions may occur for any given card.

Input to Phase 1 consists of the source text card images.

Output from Phase 1 consists of the internal text file (on tape), and the following internal files: error message, constant pool, pseudo-operation dictionary, FILE cards, and CONTRL cards.

At the completion of Phase 1, the Interlude and Phase 2 routines (which are assembled together) are brought into core storage, and control passes to the Phase 2 Supervisor.

Phase 2

Interlude processing occurs in the following order:

1. The FILE card file is examined. From this, the file dictionary is constructed. Any necessary $FILE cards are written at this time, and the file dictionary is printed and punched.

2. The pseudo-operation dictionary is brought into storage. It overlays the area occupied in Phase 1 by the Macro Skeleton Table. The definitions of all symbols are determined.

3. The CONTRL card file is examined. The control dictionary is constructed. It is complete, except for the inclusion of virtual symbols.

The input to Interlude routines is the pseudo-operation dictionary, FILE cards, and CONTRL cards.

Output, if any, consists of error messages.

Pass 2

Pass 2 of Phase 2 performs the final assembly pass over the internal text. Functions performed during this pass are:

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. The virtual symbols are determined and the control dictionary is completed.

5. The Cross-Reference Usage Table is prepared for the Input/Output Editor.

Following this pass, the control dictionary is processed to produce binary cards and list information, as required. Next, the error message file is examined. Skeletons of any messages issued are interpreted, and the appropriate text is sent to the listing. Finally, if called for, the cross-reference usage list is prepared.

Input to Phase 2 is the internal text file and the error message file.

Output from Phase 2 is the assembly listing and the binary deck, if specified.

Control is returned to the Processor Monitor upon completion of Phase 2.

Error Messages

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.

Following is a list of Assembler error messages in alphabetical order. The severity indication that accompanies a message in the listing of error messages varies with the nature of the error.

NAME TABLE FULL. ASSEMBLE PROGRAM IN SMALLER PARTS.

INTERNAL DICTIONARY FULL. ASSEMBLE PROGRAM IN SMALLER PARTS.

ILLEGAL BCD CHARACTER TREATED AS IF BLANK.

LOCATION FIELD FORMAT ERROR.

ILLEGAL SCAN CONDITION. ASSEMBLER OR MACHINE ERROR.

OPERATION FIELD FORMAT ERROR.

INCORRECT DUPLICATE SEQUENCE. POSSIBLE ASSEMBLER OR MACHINE ERROR.

VARIABLE FIELD TOO LONG.

INDIRECT ADDRESS NOT ALLOWED ON THIS INSTRUCTION.

SYMBOL TOO LONG. FIRST SIX CHARACTERS USED.

ILLEGAL INTERNAL CONDITION. ASSEMBLER OR MACHINE ERROR.

FLOATING-POINT OVERFLOW IN CONVERTING LITERAL OR CONSTANT.

FLOATING-POINT UNDERFLOW IN CONVERTING LITERAL OR CONSTANT.

SIGNIFICANT DIGITS LOST IN SHIFTING FIXED-POINT LITERAL OR CONSTANT.

MORE THAN ONE SIGN FOR PRINCIPAL PART OF LITERAL OR CONSTANT.

MORE THAN ONE DECIMAL POINT FOR LITERAL OR CONSTANT. ALL BUT FIRST IGNORED.

MORE THAN ONE SIGN FOR EXPONENTIAL OR BINARY-PLACE PART OF LITERAL OR CONSTANT.

MISUSE OF E OR B IN LITERAL OR CONSTANT.

DECIMAL POINT CAN OCCUR ONLY IN PRINCIPAL PART OF LITERAL OR CONSTANT.

FIELD TRUNCATED TO MAXIMUM DIGIT SIZE.

TOO MANY SUBFIELDS ON THIS CARD.

QUALIFICATION ILLEGAL ON THIS CARD.

S-VALUE INDETERMINATE. ASSEMBLER OR MACHINE ERROR.

LOCATION FIELD REQUIRED ON THIS CARD.

PRINCIPAL PSEUDO-OPERATION CANNOT BE DEFINED.

MIXED BOOLEAN EXPRESSION. LEFT BOOL USED.

'NAME.1' IS AN IMPROPERLY QUALIFIED NAME.

MACRO DEFINITION NESTING ERROR.

MACRO DEFINITION TERMINATED BY –ENOM– CARD. WITH BLANK VARIABLE FIELD.

MACRO DEFINITION TERMINATED BY –ENOM– CARD WITH WRONG NAME.

MACRO DEFINITION CARD WITHOUT NAME IGNORED.

MACRO DEFINITION CANNOT HAVE MORE THAN 63 PARAMETERS.

MISUSE OF PARENTHESES IN MACRO DEFINITION.

–ETC– CARD SHOULD FOLLOW.

PERMANENT READ ERROR ON SECOND PASS TEXT INPUT.

MACRO PARAMETER EXPANSION TABLE OVERFLOW.

MACRO PARAMETER PUSH DOWN TABLE OVERFLOW.

MACRO CALL HAS TOO MANY PARAMETERS.

END OF FILE WHILE PROCESSING SOURCE INPUT.

OPERATION CODE NOT IN DICTIONARY.

DUBIOUS USE OF * ASSUMED TO BE LOCATION COUNTER.

–CALL– OPERATION WITH BLANK VARIABLE FIELD IGNORED.

–CALL– OPERATION REQUIRES UNQUALIFIED LEGITIMATE SYMBOL AS TRANSFER POINT NAME.

MACRO GENERATION SYNCHRONIZATION FAILURE. ASSEMBLER OR MACHINE ERROR.

ILLEGAL VARIABLE FIELD CONDITION AT START OF –ETC– CARD.

QUALIFICATION SECTION TERMINATED BY –ENOQ– CARD WITH WRONG NAME.

QUALIFICATION SECTION NESTING ERROR.

QUALIFICATION SECTION TERMINATED BY –ENCQ– CARD WITH BLANK VARIABLE FIELD.

MACRO PARAMETER SUBSTITUTION CANNOT HAVE MORE THAN 61 CHARACTERS.

EXTERNAL NAME 'NAME.1' SUPPLIED FOR THIS CDICT ENTRY.

INCORRECT FORM OF VARIABLE FIELD ON –CONTRL– OR –FILE– CARD.

–FILE– CARD MUST HAVE SYMBOL IN LOCATION FIELD.

EXTERNAL LABEL FOR FILE TOO LONG. FIRST 18 CHARACTERS USED.

FORMAT ERROR FOLLOWING = SIGN ON –FILE– CARD.

ADDRESS REQUIRED.

ADDRESS NOT EXPECTED.

ADDRESS NOT ALLOWED.

TAG REQUIRED.

TAG NOT EXPECTED.

TAG NOT ALLOWED.

DECREMENT NOT EXPECTED.

DECREMENT REQUIRED.

DECREMENT NOT ALLOWED.

NUMERIC FIELD CONTAINS NONNUMERIC CHARACTER. FIELD IGNORED.

INCORRECT FORM OF VARIABLE FIELD ON –DRGCRS– CARD.

THIS CARD NOT EFFECTIVE UNLESS WITHIN A MACRO DEFINITION.

INTERNAL DICTIONARY OVERFLOW WHILE PROCESSING CONTROL DICTIONARY.

CONTROL DICTIONARY ENTRY FOR 'NAME.1' ELIMINATED. IMPROPER REFERENCE.

CONTROL DICTIONARY NOT CORRECTLY PROCESSED. ASSEMBLER OR MACHINE ERROR.

THIS CARD NOT EFFECTIVE IN AN ABSMOD ASSEMBLY.

OCTAL CONSTANT CANNOT EXCEED 12 DIGITS.

OCTAL CONSTANT CONTAINS NON-OCTAL CHARACTER.

ILLEGAL USE OF –ETC– CARD.

IMPROPER PUNCTUATION FOLLOWING HOLLERITH LITERAL.

COUNT FIELD ON –BCI– CARD CANNOT EXCEED SIX DIGITS.

COUNT FIELD ON –BCI– CARD IMPROPERLY TERMINATED. COUNT OF ONE USED.

THIS CARD NOT EFFECTIVE IN A RELMOD ASSEMBLY.

SERIALIZATION GROUP ON –LEL– CARD TOO LARGE. FIRST EIGHT CHARACTERS USED.

INDEX SPECIFIED MORE THAN ONCE FOR –SAVE– OPN.

SUBFIELD IGNORED BECAUSE OF FORMAT ERROR.

INTERNAL TEXT SYNCHRONIZATION FAILURE. ASSEMBLER OR MACHINE ERROR.

ONLY FIRST –LORG– EFFECTIVE.

ONLY FIRST –LDIR– EFFECTIVE.

VARIABLE FIELD TOO COMPLEX. SIMPLIFY AND REASSEMBLE.

COUNT FIELD ON –VFD– CANNOT EXCEED SIX DIGITS.

NO PREVIOUS LOCATION COUNTER. BLANK COUNTER USED.

–IRP– IGNORED. VARIABLE FIELD ERROR.

ZERO SUPPLIED FOR REQUIRED OPERAND.

TOO MANY SUBFIELDS ON –DUP– CARD. FIRST TWO USED.

LTERATION COUNT ZERO. 'NAME.1' CARDS SKIPPED.

END OF FILE WHILE PROCESSING –DUP– CARD.

–DUP– CARD PUSH DOWN TABLE OVERFLOW.

QUALIFICATION SECTION NESTING CAPACITY EXCEEDED.

BIT COUNT TOO LARGE FOR –VFD– OR –OPVFD– SUBFIELD.

VARIABLE FIELD FORMAT ERROR FOR –IFF– OR –IFT– CARD.

ONE OR MORE QUALIFICATION SECTIONS NOT CLOSED.

ILLEGAL COUNT FIELD ON –BCI– CARD.

DECIMAL CONSTANT TOO LARGE.

BOOLEAN CONSTANT TRUNCATED TO SIX DIGITS.

TOTAL BIT COUNT FOR –OPVFD– MUST NOT EXCEED 36.

PSEUDO–OPERATION DICTIONARY FULL. ASSEMBLE PROGRAM IN SMALLER PARTS.

'NAME.1' IS AN UNDEFINED SYMBOL.

BAD ADJECTIVE CODE USED FOR –OPD– OR –OPVFD–.

MACRO SKELETON TABLE OVERFLOW. NO MORE DEFINITIONS ACCEPTED.

VIRTUAL CANNOT APPEAR IN TAG.

ILLEGAL OPERAND IN THE VARIABLE FIELD.

THIS INSTRUCTION CANNOT HAVE AN ASSOCIATED SYMBOL.

ILLEGAL TERMINATOR FOR DECIMAL CONSTANT.

ILLEGAL SUBFIELD IGNORED FOR –SAVE– OPN.

NAME SUPPLIED FOR –SAVE– CPN.

INVALID INDEX SPECIFIED FOR –SAVE– OPN.

I.D OR E OCCURRED MORE THAN ONCE FOR –SAVE– OPN.

ILLEGAL OPERAND IN BOOLEAN FIELD.

ILLEGAL QUALIFICATION USAGE ON THIS CARD.

–END– SHOULD NOT OCCUR IN RANGE OF A –DUP–.

DECREMENT MUST BE CONSTANT AND CANNOT EXCEED 'NAME. 1' BITS.

–END– SHOULD NOT OCCUR IN MACRO DEFINITION.

NON-NUMERIC CHARACTER IN 'ID' FIELD OF –CALL–.

SUBFIELD 'NAME. 1' FORMAT ERROR.

5 SUBFIELDS REQUIRED ON THIS CARD. COMMAS SUPPLIED.

THIS INSTRUCTION IS NOT EXPANDED AS A MACRO.

'*' COMMENTS CARD IGNORED WITHIN MACRO DEFINITION.

LOCATION FIELD FORMAT ERROR. FIELD IGNORED.

OPERATION CODE REDEFINED.

DUBIOUS USE OF 'NAME. 1' ON –FILE– CARD.

MISUSE OF PARENTHESES ON –CALL– OPERATION.

CONTROL DICTIONARY TABLE OVERFLOW.

–END– CARD SHOULD FOLLOW.

VIRTUAL CANNOT APPEAR IN –END– CARD.

FORTRAN IV Compiler Information

The 7090/7094 FORTRAN IV Compiler (IBFTC) is a group of subroutines arranged in a hierarchy of control. Together, these subroutines are to be used as a single closed subroutine by the Processor Monitor. The function of the FORTRAN IV Compiler is to translate a 7090/7094 FORTRAN IV source program into MAP language.

Input/output functions are provided outside of the scope of the FORTRAN IV Compiler. The following subroutines are required by the FORTRAN Compiler: a subroutine to furnish the source program statements one line at a time; a subroutine to accept the output of the FORTRAN IV Compiler one line at a time and to accept diagnostic messages concerning compilation; a subroutine to provide for input/output handling of one work file during compilation.

Control

A compilation may be effected by the Processor Monitor by executing a call to FORTRAN Compiler Control (FCC). FORTRAN Compiler Control performs the compilation by calling the following in succession:

1. PH1 (Phase 1), which reduces the source program into tabular form and analyzes mathematical expressions.
2. SA (Storage Allocator), which compiles storage allocation pseudo-operations for the object program.

3. ALT (Alternator), which completes the compilation of the object program by alternately calling indexing and procedural compiling routines.

Phase One

Phase One is the first section of the FORTRAN IV Compiler to be called by FORTRAN Compiler Control. The function of Phase One is to translate the source program into a series of correlated table entries. Phase One control initiates processing for each source statement until the end of the source program is reached. After the END statement has been processed, control is returned to FORTRAN Compiler Control. Figure 18 illustrates Phase One.

ORGANIZATION

Phase One comprises a set of closed subroutines. Each subroutine is called to process a specific source statement. In addition, one processor may call upon another to accomplish its designated function. If control is passed from one routine to another, it must be passed back through the same chain to the originally called processor.

GENERAL PROCESSING PROCEDURE

Initially, a source statement, with all of its continuation cards (if any), is collected. The aggregate source statement is then scanned to determine its type: arithmetic or nonarithmetic. Control is passed to the arithmetic processor if it has been determined that the statement is arithmetic. Otherwise, further analysis must be performed in order to determine which of the nonarithmetic statement processors should be called.

Once a specific statement processor has been called, it retains control until the end of the statement is encountered. Control is returned to Phase One Control when the end of statement is reached through the Classification routine. However, if the end-of-segment flag is recognized, control is returned to the Classification routine so that the next segment can be classified and processed. The individual routines assign internal formula numbers whenever it is deemed necessary.

The entire processing procedure continues until Phase One Control either determines that there are no more source statements to be processed or that a

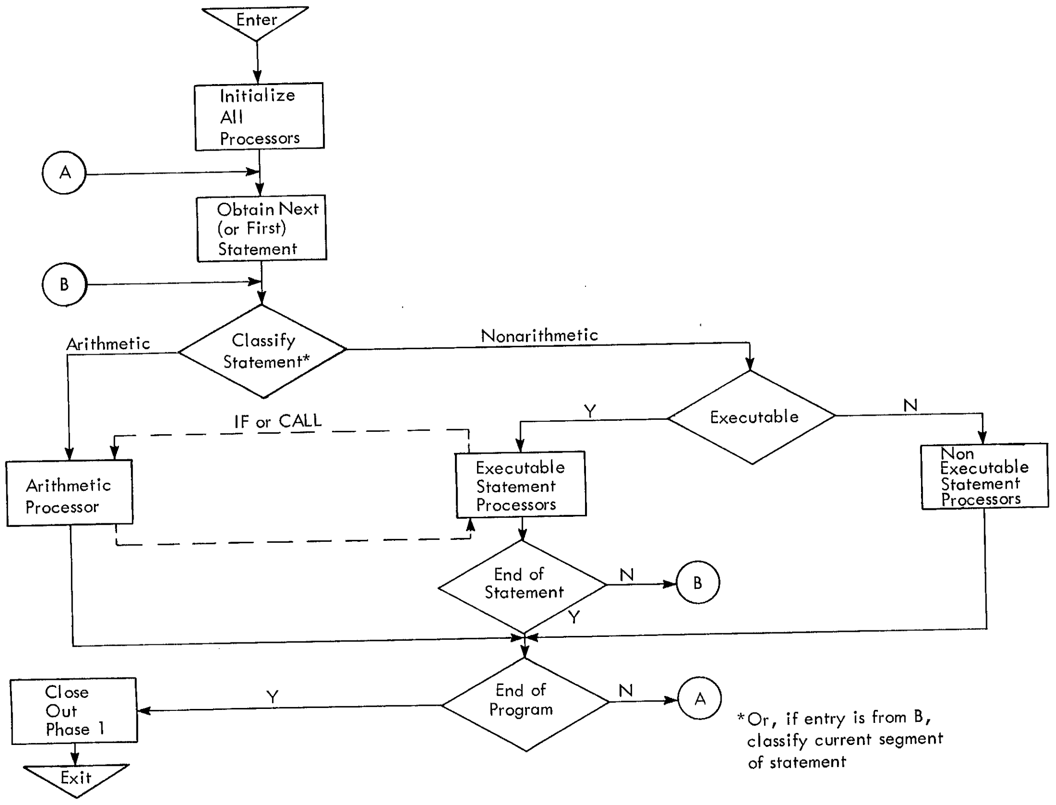

Enter
Initialize All Processors
A
Obtain Next (or First) Statement
B
Arithmetic
Classify Statement*
Nonarithmetic
Executable
Y
N
IF or CALL
Arithmetic Processor
Executable Statement Processors
Non Executable Statement Processors
End of Statement
N
B
Y
Close Out Phase 1
Y
End of Program
N
A
*Or, if entry is from B, classify current segment of statement
Exit

Figure 18. Phase One Flow Chart

condition has arisen that precludes further processing. When either terminal condition occurs, control will be returned to FORTRAN Compiler Control.

Storage Allocation

1. The Storage Allocator (SA) generates MAP pseudo-operations to reserve storage in the object for formats and all variables, in accordance with FORMAT, COMMON, EQUIVALENCE, and TYPE statements in the source program.

2. The Storage Allocator generates constants, which are loaded into reserved storage locations, in accordance with DATA statements in the source program.

3. The Storage Allocator generates EQU pseudo-operations to equate external formula numbers with internal formula numbers in the object program, replaces external formula numbers with internal formula numbers in the BRANCH table, and produces the BRADD table, a reordered version of the BRANCH table.

The following two routines are required to use the Storage Allocator:

MAP Code Generator
FCC Table Routines

ORGANIZATION

The Storage Allocator is divided into six main executive sections under the control of Storage Allocation Control (SAC). This is shown in Figure 19.

SAC: Performs necessary initialization, puts out standard location-counter control pseudo-operations, calls subsidiary sections as necessary, and performs general control operations.

FORMAT: Reduces FORMAT Table entries to BCI pseudo-operations for assembly.

COMMON: Reduces COMMON Table entries to BSS pseudo-operations under appropriate control sections for assembly. Preserves information for use by EQUIT, if necessary.

EQUIT: Performs the following functions:

1. Produces independent, nonredundant strings of equivalent variables (and associated addends) in the REQUIT Table, from the lists of equivalent variables (and addends) in the EQUIT Table.

2. Processes the REQUIT Table, string by string. Determines the base of each string; assigns an appropriately-sized storage block to the base for assembly; assigns all other members of the string relative to the base (using EQU) for assembly.

DLIST: Generates constants, which are loaded into assigned storage, according to the DLIST, DLIT, and DDO Tables.

TEIFNO: Equates external formula numbers to internal formula numbers for assembly. Replaces external formula numbers in BRANCH Table by internal formula numbers. Generates the reordered BRANCH Table, BRADD.

NAMTAB: Allocates storage to variables not appearing in COMMON or EQUIVALENCE statements.

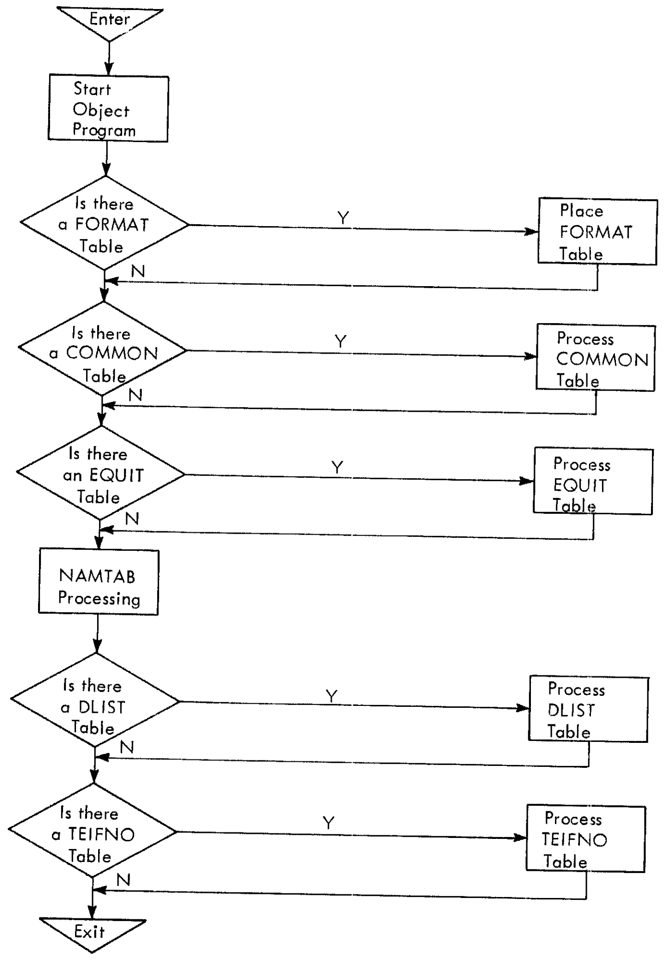


Figure 19. Storage Allocation Flow Chart

Alternator (ALT)

The Alternator is the controlling routine for compilation of the executable portion of the object program. The Alternator requires TABLE X entries one at a time. It is determined whether or not indexing conditions are met for each TABLE X entry; if so, the Indexer (IND) is called to compile indexing instructions. It is then determined whether Main Compiling conditions are met; if so, the Main Compiler (MC) is called to compile instructions for an executable statement.

Indexing conditions are:

1. The beginning of a non-DO basic block, e.g., a stretch of program containing neither DOS nor transfers into its range.

2. The beginning of a DO nest, e.g., the DO statement corresponding to the outermost DO of a nest.

Main Compiling conditions occur at every TABLE X entry except DO, DO END, and END.

When the Indexer is called for an indexing condition, it immediately compiles all the indexing instructions for the appropriate range of program. Since these instructions are not normally contiguous in the object program, i.e., they are to be interspersed with main compiling instructions, IND compiles them under the control of unique symbolic location counters. By means of the BEGIN Table, ALT compiles BEGIN pseudo-operation codes at appropriate points to effectively insert these instructions in their proper positions at loading time.

Other functions performed by the Alternator are described in the following text.

CHECKING OF TRANSFERS INTO THE RANGE OF DOS

Before the TABLE X scan is begun, the Alternator passes over the TDO, BRANCH, and BRADD Tables to check for valid and invalid transfers into the ranges of DOs. Special cases of these are allowed, and index register loading and saving counters are compiled for this purpose.

PROLOGUE COMPILATION

When the END statement is encountered in TABLE X, the PROLOG Table (compiled by the Main Compiler) is scanned against the SUBARG Table, and instructions are compiled in the prologue to initialize object program references to program arguments.

GENERATED ERASABLE WORDS

After the prologue has been compiled, the Alternator scans the ERAS Table (compiled by the Main Compiler) and compiles allocation of erasable words used by object program arithmetic (and other) instructions.

Diagnostic Routine

All source program, machine, and logic errors, detected during compilation call the Diagnostic routine, which prints a message describing the error. If the diagnostic call is caused by a source program error, the message appears off-line, either following the source program statement which contained the error or after the source program, depending upon the phase of compilation in which the error was detected.

The diagnostic message consists of two parts for a source program error and three parts for a machine or logic error.

1. The first part of the message contains a description of the error, which is provided by that part of the FORTRAN IV Compiler which detected the error.

2. The Diagnostic routine provides the second part of the message which contains the type of error (source or logic), the permissiveness level value associated with the error and a description of the level of the error.

The permissiveness level value of the error is defined as follows:

LEVEL	DESCRIPTION OR PROCEDURE
1	Warning only
2	Loading suppressed
3	Assembly suppressed
4	Compilation suppressed, error scan continues
5	Compilation abandoned

In addition to the permissiveness level number, the second part of the diagnostic message contains a unique number assigned to each diagnostic.

A typical diagnostic message follows:

```
SOURCE ERROR 4, LEVEL 4, COMPILATION
SUPPRESSED, ERROR SCAN CONTINUES.
```

or

```
LOGIC ERROR 2, LEVEL 1, WARNING ONLY.
```

3. The third part of the message is provided by the Diagnostic routine and is printed only for a machine error or a logic error. It has the following form:

DIAGNOSTIC CALL FROM (symbol) AT (location) IN (prognm_1)
(prognm_1) CALLED BY (prognm_2)
(prognm_2) CALLED BY (prognm_3) etc.

where symbol is the BCD symbolic location of the call in the FORTRAN IV Compiler:

location is the octal location of the call in storage, prognm_1 is the name of the program in the FORTRAN IV Compiler which is called the Diagnostic; prognm_2 is the name of the program in the FORTRAN IV Compiler that called prognm_1; etc.

The flow trace is continued back until FORTRAN Compiler Control becomes the calling program.

COBOL Compiler Information

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).

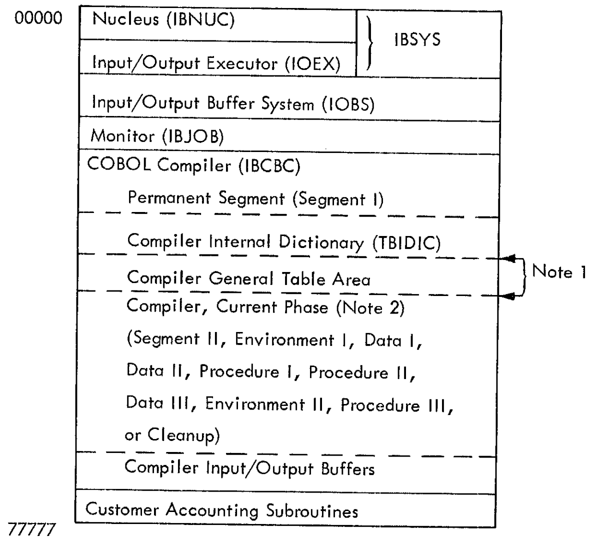
Input to the COBOL Compiler is a COBOL source program which has been put onto the System Input Unit (SYSIN1). Output from the COBOL Compiler consists of the following:

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 20. 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.



00000
Nucleus (IBNUC)
Input/Output Executor (IOEX)
IBSYS
Input/Output Buffer System (IOBS)
Monitor (IBJOB)
COBOL Compiler (IBCBC)
Permanent Segment (Segment I)
Compiler Internal Dictionary (TBIDIC)
Compiler General Table Area
Note 1
Compiler, Current Phase (Note 2)
(Segment II, Environment I, Data I, Data II, Procedure I, Procedure II, Data III, Environment II, Procedure III, or Cleanup)
Compiler Input/Output Buffers
Customer Accounting Subroutines
77777

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 20. Core Storage MAP for the 7090/7094 COBOL Compiler (IBCBC)

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.

The 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 code 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 PUTCPM subroutine, except 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 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 requires 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 phase 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 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, Data II does the following:

1. Assigns object-time base locators 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. 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 OCCURS . . . DEPENDING ON. . ., 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 phase 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 phase 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 phase 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:

```
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 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 programs 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.

Diagnostic and Error Messages

Following is a list of the COBOL Compiler diagnostic and error messages which may appear on the system output tape following the listing of the source program. There are three degrees of severity that these messages may have: W, E, and D. The severity level of the messages may be more easily remembered if the indicator characters are considered as the initial letters of the words Warning, Error, and Disaster.

A message with severity W will inhibit neither assembly nor immediate loading of the object program. A message of severity E will inhibit the immediate loading of the object program. A message of severity D will cause compilation to terminate and assembly to be inhibited.

Error Messages

Severity code (W=warning, E=error, D=disaster) and a related source program card number precede each message.

0000	ERROR MESSAGE NOT YET IN FILE.
0001	ALTER REFERENCE INCORRECT - ----- IS NOT A ----- NOTHING DONE.
0002	----- IS STRUCTURALLY INCORRECT AT THIS POINT. RERUN CLAUSE IGNORED.
0003	----- IS NOT A FILE-NAME. RERUN CLAUSE IGNORED.
0004	INVALID LITERAL USED IN EXAMINE STATEMENT.
0005	CLOSE REEL FOR ----- IS ILLEGAL SINCE FILE IS ASSIGNED TO A CARD OR SYSTEM UNIT. REEL OPTION IGNORED.

0006 USE OF ----- IN ----- STATEMENTS PREFERRED OVER ----- OR 'EQUALS'. COMPLIANCE WITH STANDARD IBM COBOL RECOMMENDED.

0007 INTEGER MUST NOT EXCEED 32767. INTEGER 1 ASSUMED.

0008 ----- CAN NOT HAVE MORE THAN 49 QUALIFIERS. EXTRA ONES DELETED.

0009 MAXIMUM NUMBER ----- OF DIFFERENT NAMES IN A SOURCE PROGRAM EXCEEDED. COMPILATION TERMINATED.

0010 ----- IS IMPROPERLY QUALIFIED. DEFINITION FORCED.

0011 ----- SHOULD NOT BE IN THE A MARGIN. B MARGIN ASSUMED.

0012 ----- IS A NAME DEFINITION AND MUST NOT BE QUALIFIED. DEFINITON FORCED.

0013 ----- IS NOT DEFINED. DEFINITION FORCED UNLESS A QUALIFIER.

0014 ----- IS IMPROPERLY QUALIFIED. NAME IS NOT UNIQUE. DEFINITION FORCED.

0015 COMPILER TABLE CAPACITY EXCEEDED. TRY SUBDIVIDING INTO SMALLER PROGRAMS FOR SEPARATE COMPILATION WITH COMBINATION AT OBJECT TIME.

0016 ----- USED AS A SUBSCRIPT IS A SIGNED EXTERNAL DECIMAL ITEM. NEGATIVE VALUES CAUSE ERRORS.

0017 USED AS A SUBSCRIPT HAS AN ALPHABETIC PICTURE. INVALID SUBSCRIPT. OBJECT PROGRAM USES SUBSCRIPT VALUE OF 1.

0018 ----- USED AS A SUBSCRIPT IS IN BCD. OBJECT-TIME CONVERSION AND/OR UNPACKING IS REQUIRED FOR SUBSCRIPTS NOT COMPUTATIONAL, SYNCHRONIZED RIGHT.

0019 IS A TYPE OF ELEMENTARY DATA ITEM THAT MAY NOT BE USED AS A SUBSCRIPT. OBJECT PROGRAM USES SUBSCRIPT VALUE OF 1.

0020 ----- USED AS A SUBSCRIPT HAS AN ALPHANUMERIC PICTURE, BUT VALUES MUST BE RESTRICTED TO INTEGERS.

0021 GROUP ITEM ----- USED AS A SUBSCRIPT. OBJECT PROGRAM USES SUBSCRIPT VALUE OF 1.

0022 ----- USED AS A SUBSCRIPT IS NOT SYNCHRONIZED RIGHT. OBJECT-TIME UNPACKING IS REQUIRED.

0023 ----- USED AS A SUBSCRIPT IS INVALID DUE TO NON-ZERO SCALING. OBJECT PROGRAM USES SUBSCRIPT VALUE OF 1.

0024 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.

0025 DISPLAY OF VARIABLE LENGTH ITEM ----- NOT PERMITTED.

0026 ----- HAS AN ILLEGAL LEVEL NUMBER. ASSUMED LEVEL NUMBER IS 49.

0027 SECTION-HEADER ----- SECTION' NOT FOLLOWED BY ----- DESCRIPTION ENTRY. SCAN RESUMED AT NEXT ----- DESCRIPTION ENTRY, SECTION, OR DIVISION.

0028 SECTIONS IN THE DATA DIVISION MUST ----- SCAN RESUMED AT NEXT SECTION, OR DIVISION.

0029 COBOL WORD 'SECTION' MISSING. BEGINNING OF ----- SECTION ASSUMED BY COMPILER.

0030 ----- DESCRIPTION ENTRY ENCOUNTERED. BEGINNING OF ----- SECTION ASSUMED BY COMPILER.

0031 ----- IS UNRECOGNIZABLE. SCAN RESUMED AT NEXT DATA DESCRIPTION ENTRY, SECTION, OR DIVISION.

0032 ENVIRONMENT ----- NOT FOLLOWED BY ----- SCAN RESUMED AT NEXT PARAGRAPH, SECTION, OR DIVISION.

0033 I-O-CONTROL PARAGRAPH NOT FOLLOWING FILE-CONTROL PARAGRAPH IS IGNORED.

0034 ----- IS UNRECOGNIZABLE. SCAN RESUMED AT NEXT PARAGRAPH, SECTION, OR DIVISION.

0035 ENVIRONMENT PARAGRAPH-NAME ENCOUNTERED. BEGINNING OF ----- SECTION ASSUMED BY COMPILER.

0036 ----- IN THE ENVIRONMENT DIVISION MUST NOT BE REPEATED. SCAN RESUMED AT NEXT PARAGRAPH, SECTION, OR DIVISION.

0037 ----- PARAGRAPH APPEARS ILLEGALLY IN ----- SECTION. SCAN RESUMED AT NEXT PARAGRAPH, SECTION, OR DIVISION.

0038 COBOL WORD 'SECTION' MISSING. BEGINNING OF ----- SECTION ASSUMED BY COMPILER.

0039 ----- IS NOT A FILE-NAME. FD ENTRY IGNORED.

0040 'FILLER' NOT PERMITTED AS FILE-NAME. FD ENTRY IGNORED.

0041 ----- HAS NOT APPEARED IN A SELECT ENTRY IN ENVIRONMENT DIVISION. FD ENTRY IGNORED.

0042 REDUNDANT FD ENTRY ----- IGNORED.

0043 REDUNDANT FD ENTRY ----- IGNORED. ONLY ONE FD ENTRY MAY DESCRIBE A SET OF RENAMED SELECT ENTRIES.

0044 ----- IS UNIDENTIFIABLE. REMAINDER OF CLAUSE IS IGNORED.

0045 MULTIPLE ----- CLAUSES IN FD ENTRY -----. FIRST ONE RETAINED.

0046 FILE ----- IS ASSIGNED TO A CARD OR SYSTEM UNIT. OPTIONS SPECIFIED IN CLOSE STATEMENT ARE IGNORED.

0047 FD ENTRY ----- NOT TERMINATED BY A PERIOD. CONDITION IGNORED.

0048 COPY OPTION NOT IMPLEMENTED. FD ENTRY IGNORED.

0049 ----- SPECIFICATION IN ----- CLAUSE IS NOT AN UNSIGNED INTEGER. CLAUSE IGNORED.

0050 ----- SPECIFICATION IN ----- CLAUSE IS NOT AN UNSIGNED INTEGER. REMAINDER OF CLAUSE RETAINED.

0051 ----- IS GREATER THAN -----. FIRST VALUE USED IN DETERMINING MAXIMUM ----- SIZE.

0052 DISPLAY OF ----- TRUNCATED AFTER FIRST 72 CHARACTERS.

0053 IMPROPER LABEL CLAUSE IGNORED. COMPILER ASSUMES LABEL RECORD(S) OMITTED UNLESS VALUE CLAUSE PRESENT.

0054 ----- NOT A LABEL-DATA-NAME. REMAINDER OF VALUE CLAUSE IGNORED.

0055 ----- SPECIFICATION --- REMAINDER OF VALUE CLAUSE IGNORED.

0056 ----- LITERAL IS TOO LONG. FIRST ----- CHARACTERS WILL BE USED.

0057 ----- IS NOT A NUMERIC LITERAL AND IS IGNORED.

0058 IMPROPER RECORDING CLAUSE IGNORED. BCD, HIGH DENSITY ASSUMED.

0059 ----- IS UNIDENTIFIABLE. CLAUSE IGNORED.

0060 COMPILER ASSUMES FILE(S) ASSOCIATED WITH FD ENTRY ----- HAS LABEL RECORD SINCE VALUE OF LABEL GIVEN.

0061 LABEL CLAUSE OMITTED IN FD ENTRY -----. COMPILER ASSUMES LABEL RECORD(S) OMITTED.

0062 VALUE CLAUSE OMITTED IN FD ENTRY -----. LEGAL BUT UNUSUAL.

0063 DATA RECORDS CLAUSE OMITTED IN FD ENTRY -----. CONDITION IGNORED.

0064 ----- HAS NO FILE DESCRIPTION. INPUT/OUTPUT STATEMENT IGNORED.

0065 FILE ----- HAS NO RECORDS. INPUT/OUTPUT STATEMENT IGNORED.

0066 DATA ITEM ----- INVALID AS AN ARGUMENT IN 'EXAMINE' STATEMENT OR CLASS TEST. STATEMENT DELETED.

0067 FILE ----- RETENTION-PERIOD SPECIFICATION IGNORED SINCE ALLOWED ONLY FOR OUTPUT FILE.

0068 NO RECORD DESCRIPTION ENTRIES FOLLOW FD ENTRY -----.

0069 RECORD ----- LISTED IN THE FD DATA RECORD(S) CLAUSE DOES NOT APPEAR IN A RECORD DESCRIPTION ENTRY. CONDITION IGNORED.

0070 RECORD ----- APPEARS IN RECORD DESCRIPTION ENTRY BUT WAS NOT LISTED IN THE FD DATA RECORD(S) CLAUSE. CONDITION IGNORED AND RECORD DESCRIPTION RETAINED.

0071 FILE ----- HAS NO ASSOCIATED FD ENTRY. ARBITRARY SPECIFICATIONS ASSUMED.

0072 FILE ----- IS ASSIGNED TO ----- AND FILE RECORDING MODE IS BINARY. UNIT NOT PERMITTED TO BE CARD AT OBJECT-TIME.

0073 CHECKPOINTS DESIGNATED TO BE WRITTEN ON FILE ----- BUT FILE IS NOT LABELED OUTPUT. CHECKPOINTS WILL BE WRITTEN ON STANDARD CHECKPOINT UNIT INSTEAD.

0074 DESIGNATION OF 'OPTIONAL' FILE, ALTHOUGH NOT STANDARD IBM COBOL, ALLOWED.

0075 FILE ----- SPECIFIED AS ----- INPUT ----- OUTPUT. ----- USAGE ASSUMED.

0076 PROCEDURE STATEMENT TO ----- FILE ----- NOT GIVEN.

0077 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.

0078 ----- FILE ----- IS ASSIGNED TO -----. UNIT NOT PERMITTED TO BE CARD AT OBJECT-TIME.

0079 FILE ----- ASSIGNED TO CARD UNIT HAS RECORD CONTAINING MORE THAN 72 CHARACTERS. MAXIMUM RECORD SIZE PROCESSED IS 72 CHARACTERS.

0080 FILE ----- IS ASSIGNED TO ----- AND MAXIMUM RECORD SIZE EXCEEDS 72 CHARACTERS. UNIT NOT PERMITTED TO BE CARD AT OBJECT-TIME.

0081 BLOCKING OF DISTINCT RECORD TYPES OF DIFFERING SIZES, WITHOUT COUNT CONTROL, IN FILE ----- IS NOT PERMITTED. FILE IS SET UNBLOCKED.

0082 BLOCK SIZE (----- COMPUTER WORDS) SPECIFIED FOR FILE ----- IS NOT A MULTIPLE OF RECORD SIZE (----- COMPUTER WORDS). BLOCK SIZE CHANGED TO ----- COMPUTER WORDS.

0083 EXCESSIVE BLOCK SIZE SPECIFIED FOR FILE -----. FILE IS SET UNBLOCKED.

0084 MAXIMUM RECORD SIZE (----- COMPUTER WORDS) EXCEEDS SPECIFIED BLOCK SIZE (----- COMPUTER WORDS) OF FILE -----. FILE IS SET UNBLOCKED.

0085 FILE ----- ASSIGNED TO SYSOU1, BUT RECORDING MODE GIVEN AS BINARY. RECORDING MODE CHANGED TO BCD.

0086 FIGURATIVE CONSTANT OR NON-NUMERIC LITERAL MUST FOLLOW 'ALL'. ALL ZERO ASSUMED IF COBOL WORD NEXT.

0087 LITERAL FOLLOWING ALL IS LIMITED TO ONE CHARACTER. THE FIRST LITERAL CHARACTER IS USED.

0088 ----- SHOULD BE FOLLOWED BY A SPACE. SPACE IS ASSUMED.

0089 ----- SHOULD NOT BE IN THE A MARGIN. B MARGIN ASSUMED.

0090 ----- SHOULD NOT BE FOLLOWED BY A SPACE. CONDITION IGNORED.

0091 INPUT/OUTPUT STATEMENT IGNORED.

0092 DOUBLE ASTERISKS (INDICATING EXPONENTIATION) SHOULD NOT BE SEPARATED BY SPACE(S). SPACE(S) IGNORED.

0093 $ IS A LEGAL CHARACTER ONLY IN THE PICTURE CLAUSE. $ DELETED.

0094 ----- IS NOT A PROCEDURE NAME. TRANSFER BYPASSING THIS STATEMENT INSERTED.

0095 -----, USED TO CONTROL A GO TO, IS NOT AN INTEGER. INTEGER PART USED.

0096 -----, USED TO CONTROL A GO TO, HAS ILLEGAL FORMAT. GO TO STATEMENT IGNORED.

0097 COMPILER BASE LOCATOR CAPACITY EXCEEDED. TRY SUBDIVIDING INTO SMALLER PROGRAMS FOR SEPARATE COMPILATION WITH COMBINATION AT OBJECT TIME.

0098 CONDITIONAL VARIABLE ----- IMPROPERLY DESCRIBED AS A REPORT, SCIENTIFIC DECIMAL, OR FLOATING POINT ITEM. X IS ASSUMED PICTURE.

0099 OCCURS CLAUSE IGNORED FOR CONDITIONAL VARIABLE -----.

0100 NOT IN 'DECLARATIVES' MODE. STATEMENT DELETED.

0101 ITEM ----- HAS NO SPECIFIED LENGTH. CONDITION IGNORED.

0102 VALUE CLAUSE OF ITEM ----- IGNORED SINCE IT IS EITHER REDEFINED, PRECEDED BY A VARIABLE LENGTH ITEM, OR IS IN THE FILE SECTION.

0103 -----, ASSOCIATED WITH REDEFINES OR OCCURS . . . DEPENDING ON . . . , IS AN IMPROPER DATA ITEM. CLAUSE IGNORED.

0104 -----, ASSOCIATED WITH OCCURS . . . DEPENDING ON . . . , IS AN IMPROPER DATA ITEM. CLAUSE IGNORED.

0105 LENGTH OF NON-NUMERIC LITERAL EXCEEDS LENGTH SPECIFIED BY SIZE OR PICTURE CLAUSE FOR -----. LOW ORDER TRUNCATION DONE.

0106 SYNCHRONIZED ITEM ----- HAS REDEFINES CLAUSE. STORAGE ASSIGNMENT MIGHT NOT BEGIN WITH THE FIRST CHARACTER POSITION OF THE REDEFINED AREA. CONDITION IGNORED.

0107 DISCREPANCY BETWEEN LEVELS OF ----- AND THE REDEFINED ITEM. DISCREPANCY IGNORED AT THIS POINT OF ANALYSIS.

0108 DATA ITEM ----- WITH REDEFINES CLAUSE NOT PRECEDED BY AN ELEMENTARY ITEM. REDEFINES IGNORED.

0109 NUMBER OF OCCURRENCES OF ITEM ----- DEPENDS ON A FOLLOWING ITEM IN THE SAME DATA GROUP. 'DEPENDING ON' CLAUSE IGNORED.

0110 NESTED REDEFINES ILLEGAL. REDEFINES CLAUSE IGNORED FOR -----.

0111 ----- WITH REDEFINES CLAUSE NOT IMMEDIATELY PRECEDED BY THE RDEFINED AREA. REDEFINES IGNORED.

0112 NON-ALPHABETIC LITERAL GIVEN FOR ALPHABETIC ITEM -----. CONDITION IGNORED.

0113 LENGTH (----- CHARACTERS) OF REDEFINING DATA FIELD HEADED BY DATA-NAME ----- IS GREATER THAN LENGTH (----- CHARACTERS) OF DATA FIELD BEING REDEFINED. DANGEROUS CONDITION IGNORED.

0114 SUBORGANIZATION OF ITEM ----- WITH OCCURS CLAUSE CONTAINS A VALUE CLAUSE. VALUE GIVEN TO FIRST ELEMENT ONLY.

0115 RECORD HEADED BY DATA ITEM ----- EXCEEDS 32767 WORDS. LENGTH MODULO 32768 USED.

0116 LIMIT (15 BITS) OF SIZE FIELD IN DICTIONARY NECESSITATES TREATING LENGTH OF ----- AS MODULO 32768.

0117 NUMBER OF DIGITS IN FIELD OF LITERAL EXCEEDS LIMIT OF 18. 18 DIGITS USED.

0118 MISUSE OF PERIOD, SIGN, OR E IN LITERAL. ILLEGAL CHARACTER(S) IGNORED.

0119 ILLEGAL CHARACTER IN LITERAL. CHARACTER IGNORED.

0120 FLOATING POINT OVERFLOW IN CONVERTING LITERAL. MAXIMUM VALUE USED.

0121 FLOATING POINT UNDERFLOW CONVERTING LITERAL. ZERO USED.

0122 ----- ----- - FROM ----- ----- TO ----- -----. ----- -----.

0123 MOVE FROM A FIGURATIVE CONSTANT TO A VARIABLE LENGTH GROUP ITEM NOT ALLOWED.

0124 MOVE FROM A FIGURATIVE CONSTANT TO AN ITEM LONGER THAN 32767 CHARACTERS NOT ALLOWED.

0125 CAUTION, GROUP LEVEL MOVE FROM ----- TO -----.

0126 CAUTION. MOVE FROM ----- TO ----- CAUSES TRUNCATION.

0127 CHARACTER LOGIC MOVE INVOLVING AN ITEM LONGER THAN 32767 CHARACTERS. NOTHING GENERATED.

0128 HYPHENATED FORM OF ----- DESIGNATION PREFERRED.

0129 OBJECT-COMPUTER DESIGNATION OVERRIDDEN BY ----- OPTION ON $IBCBC CARD.

0130 CAUTION, GROUP ITEM ----- TESTED.

0131 ----- IS STRUCTURALLY INCORRECT AT THIS POINT. SCAN RESUMED AT NEXT VERB, PERIOD, OR INFORMATION IN THE MARGIN.

0132 TOO FEW OPERANDS IN ADD STATEMENT. STATEMENT DELETED.

0133 STATEMENT REQUIRES A PROCEDURE NAME, NOT -----, AS AN ARGUMENT. STATEMENT DELETED.

0134 COBOL WORDS 'TO PROCEED TO' NOT FOUND WHERE REQUIRED IN ALTER STATEMENT. STATEMENT DELETED.

0135 DATA-NAME, NOT -----, EXPECTED AS ARGUMENT IN THIS STATEMENT. STATEMENT DELETED.

0136 COBOL WORD ----- WAS NOT FOUND WHERE REQUIRED IN THIS STATEMENT. STATEMENT DELETED.

0137 SATEMENT REQUIRES A DATA-NAME, LITERAL, OR FIGURATIVE CONSTANT, NOT -----, AS AN ARGUMENT. STATEMENT DELETED.

0138 CROSS-REFERENCE NAME TOO LONG. FIRST 6 CHARACTERS USED.

0139 STATEMENT CONTAINS TOO MANY RIGHT PARENTHESES. EXTRA PARENTHESES IGNORED.

0140 STATEMENT CONTAINS TOO FEW RIGHT PARENTHESES. COMPENSATING

PARENTHESES ADDED AT END OF STATEMENT.

0141 EXTRANEOUS 'ELSE' FOUND. 'ELSE' IGNORED. FOLLOWING TEXT TREATED AS IF 'ELSE' DID NOT EXIST.

0142 NO PARAGRAPH NAME FOUND PRECEDING 'EXIT' STATEMENT. CONDITION IGNORED.

0143 ONLY FILE-NAMES MAY BE USED AS ARGUMENTS IN 'INPUT' OR 'OUTPUT' STATEMENTS. STATEMENT DELETED.

0144 'INPUT' OR 'OUTPUT' MUST FOLLOW VERB IN AN 'OPEN' SATEMENT. STATEMENT DELETED.

0145 NUMBER OF FILES NAMED IN FILE-CONTROL EXCEEDS MAXIMUM OF 63. COMPILATION TERMINATED.

0146 ----- IS NOT -----. INPUT/OUTPUT STATEMENT IGNORED.

0147 ----- IS NOT -----. INPUT/OUTPUT STATEMENT IGNORED.

0148 ----- IS STRUCTURALLY INCORRECT AT THIS POINT. SCAN RESUMED AT BEGINNING OF NEXT SWITCH-NAME ENTRY, PERIOD, OR PROPER INFORMATION IN THE A MARGIN.

0149 ----- IS NOT A LEGAL FILE-NAME. SELECT ENTRY IGNORED.

0150 REDUNDANT SELECT ENTRY ----- IGNORED.

0151 INELIGIBLE DATA-NAME CANNOT BE USED AS AN ARGUMENT FOR THE CORRESPONDING OPTION.

0152 'RENAMING' MAY ONLY BE FOLLOWED BY A FILE-NAME. REMAINDER OF SELECT ENTRY IGNORED. ASSUMED UNIT ASSIGNMENT IS '1 TAPE-UNIT'.

0153 COBOL WORDS 'ASSIGN TO' OMITTED IN SELECT ENTRY -----. ASSUMED UNIT ASSIGNMENT IS '1 TAPE-UNIT'.

0154 ----- HAS NOT BEEN DEFINED IN A SELECT ENTRY AND IS IGNORED.

0155 OPERATION IGNORED BECAUSE ----- HAS IMPROPER DATA FORMAT.

0156 TRANSFER BYPASSED BECAUSE ----- IS NOT A STATEMENT OR SECTION NAME.

0157 ILLEGAL SENTENCE STRUCTURE. NOTHING DONE.

0158 OPERATION IGNORED BECAUSE ILLEGAL USE OF FIGURATIVE CONSTANT.

0159 INCOMPLETE STATEMENT DELETED.

0160 CONDITIONAL EXPRESSION TEST CAPACITY EXCEEDED. REWRITE AS TWO OR MORE SEPARATE EXPRESSIONS WITH A MAXIMUM OF 18 OPERATORS. ILLEGAL SENTENCE STRUCTURE. NOTHING DONE.

0161 ATTEMPTED DIVISION BY ZERO BYPASSED. RESULT TAKEN TO BE ZERO.

0162 CANNOT USE VARIABLE LENGTH ITEMS FOR COMPARISON. NOTHING DONE.

0163 DOWNSCALE GENERATED WHICH LOSES ALL SIGNIFICANT DIGITS.

0164 UPSCALE MAY CAUSE HIGH ORDER TRUNCATION FOR STORE INTO -----.

0165 ----- IS STRUCTURALLY INCORRECT AT THIS POINT. SCAN RESUMED AT BEGINNING OF NEXT SELECT ENTRY, PERIOD, OR PROPER INFORMATION IN THE A MARGIN.

0166 ----- IS STRUCTURALLY INCORRECT AT THIS POINT. REMAINDER OF SWITCH-NAME ENTRY IGNORED.

0167 ----- IS NOT A LEGAL MNEMONIC-NAME. REMAINDER OF SWITCH-NAME ENTRY IGNORED.

0168 ----- IS NOT A LEGAL CONDITION-NAME. REMAINDER OF SWITCH-NAME ENTRY IGNORED.

0169 TERMINATION OF LITERAL FORCED AT END OF CARD.

0170 MNEMONIC ----- NOT UNIQUE. CONDITION IGNORED.

0171 REDUNDANT SWITCH-NAME ENTRY FOR KEY ----- IGNORED.

0172 A SPACE SHOULD SEPARATE A SUBSCRIPTED NAME FROM THE FOLLOWING LEFT PARENTHESIS. SPACE IS ASSUMED.

0173 ILLEGAL SUBSCRIPT STRUCTURE. SCAN RESUMED AT NEXT VERB, PERIOD, OR INFORMATION IN THE A MARGIN.

0174 SUBSCRIPT INTEGER MUST NOT EXCEED 32767. INTEGER 1 ASSUMED.

0175 SUBSCRIPT COUNT EXCEEDS 3. SCAN RESUMED AT NEXT VERB, PERIOD, OR INFORMATION IN THE A MARGIN.

0176 SUBSCRIPT MISSING AFTER LEFT PARENTHESIS. SCAN RESUMED AT NEXT VERB, PERIOD, OR INFORMATION IN THE A MARGIN.

0177 DIVISION NAME SHOULD BE FOLLOWED BY THE WORD DIVISION AND A PERIOD. CONDITION IGNORED.

0178 $CBEND CARD IS MISSING ON SOURCE DECK. CONDITION IGNORED.

0179 ----- IS AN UNRECOGNIZABLE ITEM ON CARD. COMPILER SKIPS TO NEXT DIVISION.

0180 DIVISIONS MUST BE IN ORDER AND NOT DUPLICATED. COMPILER SKIPS TO NEXT DIVISION.

0181 COBOL COMPILER DOES NOT OBEY THE USE OF XR4, XR5, OR XR6 ON $IBCBC CARD. XR3 IS ASSUMED.

0182 ----- IS ILLEGAL ITEM ON $IBCBC CARD. CONDITION IGNORED.

0183 DECK NAME IS MISSING ON $IBCBC CARD. CONDITION IGNORED.

0184 ILLEGAL CHARACTER IN COLUMN 7. SPACE IS ASSUMED.

0185 CONTINUATION CHARACTER MUST NOT BE USED WITH AN OCCUPIED A MARGIN. CONTINUATION CHARACTER IGNORED.

0186 NON-NUMERIC LITERAL CONTINUATION MUST BEGIN WITH A QUOTE. QUOTE ASSUMED PRECEDING FIRST NON-SPACE CHARACTER.

0187 CARD SEQUENCE ERROR IN COLUMNS 1-6. CONDITION IGNORED.

0188 REDUNDANCY ON SYSTEM INPUT UNIT. CONDITION IGNORED.

0189 21 PERMANENT READ REDUNDANCIES ON SYSTEM INPUT UNIT. COMPILATION TERMINATED.

0190 NON-NUMERIC LITERAL LONGER THAN 120 CHARACTERS OR NAME LONGER THAN 30 CHARACTERS TRUNCATED.

0191 ILLEGAL CHARACTER ON CARD DELETED.

0192 PRIMARY AND SECONDARY UNITS ASSIGNED TO FILE ----- CONFLICT. SECONDARY UNIT ASSIGNMENT IGNORED.

0193 ILLEGAL ----- UNIT ASSIGNED TO FILE -----. -----

0194 MULTIPLE REEL OPTION FOR FILE ----- OMITTED WHERE REQUIRED BUT IS ASSUMED.

0195 ----- IS STRUCTURALLY INCORRECT AT THIS POINT. SCAN RESUMED AT BEGINNING OF NEXT RERUN CLAUSE, PERIOD, OR PROPER INFORMATION IN THE A MARGIN.

0196 ----- IS UNRECOGNIZABLE IN PROBABLE MULTIPLE REEL OPTION. MULTIPLE REELS ASSUMED.

0197 CARD UNIT NOT ALLOWED AS SECONDARY UNIT ASSIGNED TO FILE -----. SECONDARY UNIT ASSIGNMENT IGNORED.

0198 ----- IS NOT -----. INPUT/OUTPUT STATEMENT IGNORED.

0199 ARGUMENT NUMBER ----- MAY NOT APPEAR IN A DISPLAY STATEMENT. SPACE ASSUMED INSTEAD.

0200 ----- IS NOT -----. INPUT/OUTPUT STATEMENT IGNORED.

0201 ----- DEFINED IN MORE THAN ONE OUTPUT FILE. INPUT/OUTPUT STATEMENT IGNORED.

0202 ----- HAS AN ILLEGAL PICTURE. ----- IS ASSUMED PICTURE.

0203 UNIDENTIFIABLE WORD ----- IN DATA DESCRIPTION. WORD OR CLAUSE IGNORED.

0204 PREVIOUS DATA DESCRIPTION NOT TERMINATED BY A PERIOD. PERIOD ASSUMED AND PROCESSING OF ----- BEGUN.

0205 SIZE, POINT, SIGNED, OR EDITING CLAUSES IGNORED IN FAVOR OF PICTURE IN -----.

0206 MULTIPLE ----- CLAUSES IN ----- DATA DESCRIPTION. FIRST ONE RETAINED.

0207 JUSTIFIED CLAUSE IN DESCRIPTION OF ----- IGNORED. THIS FEATURE NOT IMPLEMENTED.

0208 COMPILER FORCED TO ASSUME ----- IS A GROUP ITEM DUE TO ERROR IN SUBSEQUENT LEVEL NUMBER.

0209 ELEMENTARY LEVEL CLAUSES IN DESCRIPTION OF GROUP ITEM ----- IGNORED (I.E. VALUE, SIGNED, POINT, SYNCHRONIZED, EDITING, OR PICTURE).

0210 BINARY COMPUTATIONAL USAGE OF ----- INCOMPATIBLE WITH BCD RECORDING MODE FOR THIS FILE.

0211 ----- OF GROUP ITEM ----- IGNORED DUE TO CONFLICT WITH A HIGHER ORDER GROUP.

0212 LEVEL OF ----- CONFLICTS WITH THE PRECEDING LEVEL NUMBER. CONDITION IGNORED.

0213 ILLEGAL CLAUSE(S) IN DESCRIPTION OF FLOATING POINT ITEM ----- IGNORED.

0214 ILLEGAL CLAUSE(S) IN DESCRIPTION OF SCIENTIFIC DECIMAL ITEM ----- IGNORED.

0215 ILLEGAL PICTURE OF SCIENTIFIC DECIMAL ITEM -----. +99999999E+99 IS ASSUMED PICTURE.

0216 ILLEGAL PICTURE (OR NEITHER PICTURE NOR LEGAL SIZE GIVEN) FOR ----- DECIMAL ITEM -----. 999999 IS ASSUMED PICTURE.

0217 ALPHABETIC OR ALPHANUMERIC CLASS SPECIFIED FOR ----- IGNORED SINCE ITEM IS INTERNAL DECIMAL.

0218 PICTURE OF ALPHANUMERIC ITEM ----- CONTAINS A MIXTURE OF A'S AND 9'S – TREATED AS ALL X'S.

0219 ILLEGAL USAGE OR CLASS CLAUSE (OR BLANK WHEN ZERO) IN DESCRIPTION OF ALPHANUMERIC ITEM -----. CLAUSE IGNORED IN FAVOR OF PICTURE.

0220 NEITHER PICTURE NOR SIZE CLAUSE GIVEN FOR NON-REPORT DISPLAY ITEM -----. X IS ASSUMED PICTURE.

0221 ALPHABETIC CLASS SPECIFIED FOR ----- IGNORED SINCE ITEM IS EXTERNAL DECIMAL.

0222 MULTIPLE CONTIGUOUS COMMAS IN PICTURE REPORT OF ITEM ----- CHANGED TO A SINGLE COMMA.

0223 LEVEL 77 ITEM ----- MAY NOT HAVE OCCURS.

0224 PICTURE OF REPORT ITEM ----- HAS ILLEGAL USE OF SCALING CHARACTER P. +++++++ IS ASSUMED PICTURE.

0225 PICTURE OF REPORT ITEM ----- HAS SCALING CHARACTER P EMBEDDED ILLEGALLY BETWEEN NUMERIC CHARACTER POSITIONS. +++++++ IS ASSUMED PICTURE.

0226 ILLEGAL USE OF V OR POINT IN PICTURE OF REPORT ITEM -----. +++++++ IS ASSUMED PICTURE.

0227 IMPROPER CHARACTER INTERRUPTS STRING OF + OR – OR $ IN PICTURE OF REPORT ITEM -----. +++++++ IS ASSUMED PICTURE.

0228 NO DIGIT POSITIONS IN PICTURE OF REPORT ITEM -----. +++++++ IS ASSUMED PICTURE.

0229 ILLEGAL USAGE OR CLASS CLAUSE(S) IGNORED IN FAVOR OF PICTURE OF REPORT ITEM -----.

0230 COMMA ILLEGALLY TO RIGHT OF POINT IN PICTURE OF REPORT ITEM -----. +++++++ IS ASSUMED PICTURE.

0231 ILLLEGAL MIXTURE OF DIGIT POSITION CHARACTERS (9 Z *) AFTER POINT IN PICTURE OF REPORT ITEM -----. +++++++ IS ASSUMED PICTURE.

0232 ILLEGAL USE OF COMMA IN PICTURE OF REPORT ITEM -----. +++++++ IS ASSUMED PICTURE.

0233 ILLEGAL USE OF $ IN PICTURE OF REPORT ITEM -----. +++++++ IS ASSUMED PICTURE.

0234 ILLEGAL MIXTURE OR ORDER OF DIGIT POSITION CHARACTERS IN PICTURE OF REPORT ITEM -----. +++++++ IS ASSUMED PICTURE.

0235 LEVEL NUMBER OF ----- FOLLOWING FD ENTRY CHANGED TO 01 TO DESCRIBE RECORD.

0236 DATA ITEM ----- IS UNDER INFLUENCE OF INCONSISTENT USAGE AND CLASS CLAUSES. DETERMINING HIERARCHY IS PICTURE, USAGE, CLASS.

0237 REDEFINES DESCRIBING ----- NOT FOLLOWED BY A PREVIOUSLY DEFINED DATA-NAME. CLAUSE IGNORED.

0238 ILLEGAL REDEFINITION IGNORED FOR FILE RECORD (01 LEVEL) NAMED -----.

0239 NUMBER OF OCCURRENCES OF ----- IS ILLEGAL. COMPILER ASSUMES OCCURS EXACTLY 100 TIMES.

0240 MAXIMUM NUMBER OF OCCURRENCES OF ----- IS ILLEGAL. COMPILER ASSUMES OCCURS AT MOST ----- TIMES.

0241 ERROR IN OCCURS . . . DEPENDING ON CLAUSE IN DESCRIPTION OF -----. COMPILER ASSUMES OCCURS EXACTLY 100 TIMES, IGNORING QUANTITY ITEM.

0242 COMPILER IGNORES ILLEGAL CLAUSES IN DESCRIPTION OF LEVEL 88 CONDITION ----- (ONLY VALUE IS ALLOWED).

0243 LEVEL 88 CONDITION ----- LACKS MANDATORY VALUE CLAUSE.

0244 LEVEL 88 CONDITION ----- APPEARS ILLEGALLY IN CONSTANT SECTION.

0245 LEVEL 88 CONDITION ----- NOT PRECEDED BY VALID ELEMENTARY ITEM.

0246 ILLEGAL POINT OR SIGNED CLAUSE CLAUSE IN DESCRIPTION OF NON-REPORT DISPLAY ITEM -----.

0247 NEITHER PICTURE NOR SIZE CLAUSE GIVEN FOR REPORT ITEM -----. ******* IS ASSUMED PICTURE.

0248 LEAVING OPTION OF EDIT CLAUSE EXCEEDS SIZE OF REPORT ITEM -----. CLAUSE IGNORED.

0249 ----- HAS VALUE CLAUSE TOGETHER ILLEGALLY WITH OCCURS OR REDEFINES. VALUE ACCEPTED IF OCCURS–APPLIED TO FIRST ELEMENT.

0250 PICTURE OF REPORT ITEM ----- IS ILLEGAL. +++++++ IS ASSUMED PICTURE.

0251 LEVEL 77 ITEM ----- APPEARS IN FILE SECTION. INVALID DATA ORGANIZATION RESULTS.

0252 ILLEGAL CLAUSE(S) DESCRIBING ----- IGNORED. LENGTH 1, VALUE OF R.M. ASSIGNED BY COMPILER. ONLY SYNCHRONIZATION, IF ANY, RETAINED.

0253 SIZE OF ----- GIVEN AS O. COMPILER ASSUMES SIZE IS 6.

0254 ----- CAN NOT BE SUBSCRIPTED. SCAN RESUMED AT NEXT VERB, PERIOD, OR INFORMATION IN THE A MARGIN.

0255 ILLEGAL DESIGNATION OF SIGN CONVENTION IN PICTURE OF REPORT ITEM -----. +++++++ IS ASSUMED PICTURE.

0256 NON-NUMERIC LITERAL VALUE OF NUMERIC ITEM ----- IGNORED.

0257 ----- IS NOT A LITERAL. CLAUSE IGNORED.

0258 BINARY RECORDING MODE SPECIFICATION OF FILE ----- ASSSIGNED TO CARD UNIT IS NOT PERMITTED.

0259 PERMANENT READ ERROR DURING PROCEDURE INSTRUCTION GENERATION PHASE. COMPILATION IS SUSPECT.

0260 SENTENCE LENGTH EXCEEDS COMPILER CAPACITY. SUGGEST SUBDIVIDING SENTENCE INTO SMALLER COMPONENTS.

0261 ----- FILE ----- NOT PERMITTED AS ARGUMENT IN 'USE' OPTION ----- STATEMENT.

0262 ----- AND ----- HAVE NO CORRESPONDING SUBFIELDS. NO ACTION STATEMENTS GENERATED FOR THIS PAIR.

0263 ILLEGAL CONDITIONAL EXPRESSION IN TEXT. EXPRESSION IGNORED.

0264 ----- CANNOT BE USED AS AN ARGUMENT FOR THE CORRESPONDING OPTION.

0265 CORRESPONDING FIELDS OF ----- AND ----- OVERLAP.

0266 ILLEGAL ARITHMETIC PHRASE , ENDING WITH AN OPERATOR OTHER THAN RIGHT PARENTHESIS. PHRASE DELETED.

0267 ALTER AT ----- DISALLOWED SINCE IT IS NOT A SINGLE GO TO SENTENCE.

0268 OPERAND TABLE OVERFLOW TRANSLATING EXPRESSION. STATEMENT DELETED.

0269 ERRONEOUS PARENTHESIZATION IGNORED. TRANSLATION CONTINUES ARBITRARILY.

0270 COMPILER ALLOWS ONLY 20 CONSECUTIVE IMPLIED BOOLEAN OPERATORS. CONDITIONAL EXPRESSION DELETED SINCE MAXIMUM EXCEEDED.

0271 ARITHMETIC PHRASES IN CONDITIONAL EXPRESSIONS MAY NOT CONSIST OF MORE THAN 500 OPERATORS AND OPERANDS. EXPRESSION DELETED SINCE LIMIT EXCEEDED.

0272 PERFORM STATEMENT STRUCTURALLY INCORRECT. STATEMENT DELETED.

0273 'USING' MUST BE FOLLOWED BY THE NAME OF A FIXED-LOCATION DATA-ITEM. STATEMENT DELETED.

0274 ----- IS UNIDENTIFIABLE. REMAINDER OF CLAUSE IGNORED.

0275 DECK NAME IN 'CALL' STATEMENT MUST BE ENCLOSED IN QUOTES. STATEMENT DELETED.

0276 OCCURS CLAUSE DESCRIBING RECORD ----- IN THE FILE SECTION IGNORED.

0277 ----- OF FILE ----- ASSIGN TO ----- NOT PERMITTED.

0278 ILLEGAL USE OF UNALTERABLE 'GO TO' STATEMENT. 'GO TO' STATEMENT DELETED.

0279 MACHINE OR COMPILER ERROR. COMPILATION IS INCOMPLETE.

0280 NOTE . . . FILE-NAME CHANGED FOR INTERNAL PURPOSES TO -----.

0281 TOO FEW SUBSCRIPTS GIVEN FOR -----. COMPILER ASSUMES MISSING LEFTMOST SUBSCRIPTS TO BE 1.

0282 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.

0283 FILE ----- ASSIGNED TO -----, BUT FILE IS NOT -----. ----- USAGE ASSUMED.

0284 729-MODEL NO. ----- ASSIGNED TO FILE -----, NOT ACCEPTABLE TO LOADER, CHANGED BY COMPILER TO -----.

0285 'USE' NOT PRECEDED BY SECTION-NAME.

0286 'UPON' MAY ONLY BE FOLLLOWED BY IBJOB STANDARD MNEMONIC-NAME 'SYSOU1'. SYSOU1 ASSUMED.

0287 'COLLATE-COMMERCIAL' SHOULD NOT APPEAR IN ENVIRONMENT DIVISION. COLLATING SEQUENCE ASSUMED COMMERCIAL UNLESS 'BINSEQ' APPEARS ON ON $IBCBC CARD.

0288 STANDARD IBM COBOL WORDING 'EVERY BEGINNING OF REEL' PREFERRED IN RERUN CLAUSE AND IS ASSUMED.

0289 COMPILER ----- COUNT CONTROL CONVENTION ----- FILE -----.

0290 COMPILER ----- COUNT CONTROL CONVENTION ----- FILE ----- UNLESS ----- IS ASSIGNED TO A CARD UNIT OBJECT-TIME.

0291 'ACCEPT' MAY ONLY BE FOLLOWED BY A DATA-NAME. NOTHING DONE.

0292 DATA-NAME ----- REQUIRING CONVERSION, EDITING, OR DEFINITION MAY NOT APPEAR IN AN ACCEPT STATEMENT. NOTHING DONE.

0293 ALL CHARACTERS ACCEPTED FOR ----- MUST BE NUMERIC.

0294 LENGTH OF ----- NOT BOTH CONSTANT AND LESS THAN 73 CHARACTERS. NOTHING DONE.

0295 SOURCE-COMPUTER IS IMPROPERLY SPECIFIED. IBM-7090 ASSUMED.

0296 'WITHOUT' MUST BE FOLLOWED BY COBOL WORDS 'COUNT CONTROL' IN RECORDING CLAUSE. NOTHING DONE.

0297 DATA DIVISION-HEADER NOT FOLLOWED BY SECTION-NAME. SCAN RESUMED AT NEXT DATA DESCRIPTION ENTRY, SECTION, OR DIVISION.

0298 LEVEL FD SHOULD APPEAR IN THE A MARGIN. A MARGIN ASSUMED.

0299 'FROM' MAY ONLY BE FOLLOWED BY IBJOB STANDARD MNEMONIC-NAME 'SYSIN1'. SYSIN1 ASSUMED.

0300 PICTURE SHOULD BE USED TO DESCRIBE REPORT ITEMS INSTEAD OF EDITING CLAUSES (ZERO SUPPRESS, CHECK PROTECT, OR FLOAT DOLLAR SIGN) IN ORDER TO COMPLY WITH STANDARD IBM COBOL.

0301 'DECLARATIVES' MUST BE AT BEGINNING OF PROCEDURE DIVISION. STATEMENT DELETED.

0302 FILE ----- ASSOCIATED WITH REDUNDANT 'USE' STATEMENT. FIRST ONE RETAINED.

0303 REDUNDANT 'USE' STATEMENT. FIRST ONE RETAINED.

0304 QUANTITY ITEM ----- SHOULD NOT BE USED WITH 'REPLACING' OPTION IN 'EXAMINE' STATEMENT. CONDITION IGNORED.

0305 ILLEGAL ARGUMENT IN 'ON' STATEMENT. STATEMENT DELETED.

0306 CONTROL CARD ENCOUNTERED PRECEDING $CBEND CARD. END OF COBOL TEXT ASSUMED.

0307 END OF COBOL MESSAGES.

0308 ILLEGAL CONTROL SECTION NAME FOR DEBUG REQUEST. NAME IGNORED.

0309 ILLEGAL INSERTION POINT SPECIFICATION FOR DEBUG REQUEST. REQUEST WILL NOT BE EXECUTED.

0310 ----- IS AN OUT-OF-RANGE REFERENCE.

0311 ILLEGAL FORM OF VALUE FOR ----- -----. VALUE IGNORED.

0312 ----- ORDER TRUNCATION OCCURS IN GENERATION OF INITIAL VALUE FOR -----.

0313 INELIGIBLE DATA-NAME ----- IN RECEIVE OR PROVIDE STATEMENT. NOTHING GENERATED.

0314 ----- IS A TYPE OF ELEMENTARY DATA ITEM THAT MAY NOT BE USED IN 'RETURN'. STATEMENT DELETED.

Subroutine Library Information

The Subroutine Library contains a collection of relocatable subroutines for system and programmer use. These subroutines are available to the programmer through the Loader, which incorporates them, as required, into the object program at load time.

Subroutines may be added to or deleted from the Subroutine Library by using the Librarian. Information regarding modification of the Subroutine Library may be found in the section "The Librarian." All subroutines included in the Subroutine Library must be assembled by the Assembler.

The Subroutine Library is composed of the following types of subroutines:

System subroutines
FORTRAN IV subroutines
COBOL subroutines

The system subroutines, the COBOL subroutines, the FORTRAN IV input/output subroutines, and some FORTRAN IV utility subroutines are described in the following text. All other subroutines are described in the section "The Subroutine Library (IBLIB)."

System Subroutines

The following chart contains a list and descriptions of the system subroutines. Subroutines marked with an asterisk are called automatically for every object program.

SUBROUTINE	DESCRIPTION
* .IBSYS	Defines the indices 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 above IOEX.
* .LXCON	Postexecution subroutine required for all object program executions. It is normally entered at the termination of an object program execution, but it is also entered if execution is terminated by a system stop or an STR instruction. Files used by the object program are closed, thereby stopping all input/output activity. Return is made to the System Monitor.
.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 FORTRAN IOCS; .IOCSM for minimum IOCS; .IOCSM and .IOCSB for Basic IOCS; and .IOCSM, .IOCSB, and .IOCSL for Label IOCS.
.IOCSF	Contains the text for the special IOCS used by FORTRAN IV object programs and initializes the communication region required for 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	Read/write select routine.

* .FPTRP	Floating-point trap subroutine. It determines the condition that caused the 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 of times (plus 1) that messages will be written for each execution. This number is set at five (plus 1), but it can be changed by the programmer. The number in location COUNT, which is in the control section .COUNT, may be changed by addressing this control section to 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 execution.

.RAND	Provides for processing of random records on 1301 Disk Storage.

The subroutines pertaining to IOCS must be in the following order in the Subroutine Library:

.IODEF
.IOCSF
.IOCS
.IOCSM
.IOCSB
.IOCSL

FORTRAN IV Subroutines

FORTRAN Object-Time Subroutines

This section contains a description of subroutines used by FORTRAN IV object programs. The FORTRAN IV Compiler generates instructions that call the appropriate object-time subroutines from the Subroutine Library.

The following types of subroutines are available in the FORTRAN section of the Subroutine Library:

1. FORTRAN Mathematics Library
   a. Single-precision subroutines
   b. Double-precision subroutines
   c. Complex subroutines
2. FORTRAN Input/Output Library
3. FORTRAN Utility Library

The FORTRAN Mathematics Library is described in the section "The Subroutine Library (IBLIB)."

FORTRAN Input/Output Library

The FORTRAN input/output library provides subroutines that are necessary to implement the source language input/output statements. These subroutines are described in the following chart:

SUB-ROUTINE	DESCRIPTION	ENTRY POINT	DESCRIPTION
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

SUB-ROUTINE	DESCRIPTION	ENTRY POINT	DESCRIPTION
			entry point .LXSL in subroutines .LXCON, .FPTRP, and FXEM are overlaid by a call to subroutine FOUT.
FRWD	Controls the input and output of BCD records and the conversion of alphameric data.	.FRDD.	Entry point for BCD read; called for source program statement READ (unit, format) list.
		.FWRD.	Entry point for BCD write; called for source program statement WRITE (unit, format) list.
		.CDIN.	Entry point called by subroutine FRCD.
		.FCNV.	Entry point for input/output list; effects the necessary conversion for input or output list items.
		.FPRN.	Entry point for source program statement PRINT format, list.
		.FPUN.	Entry point for source program statement PUNCH format, list.
		.FRTN.	Entry point for end of list for BCD input.
		.FFIL.	Entry point for end of list for BCD output.
		.TOUT.	Entry point at which the call to entry point .FOUT. is loaded if subroutine UN06 is used by the object program.
		.FILL.	Argument for the call to entry point .FOUT.
FRWB	Controls the input and output of binary data.	.FRDB.	Entry point for binary read; called for source program statement READ (unit) list.
		.FWRB.	Entry point for binary write; called for source program statement WRITE (unit) list.
		.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.
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.
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.
FSLI	Sets up indexing for input of 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.
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 of complex data.
		.FSDO.	Entry point for output of nonsubscripted double-precision BCD arrays.
FSLBO	Controls processing of lists containing nonsubscripted 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.
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

SUB-ROUTINE	DESCRIPTION	ENTRY POINT	DESCRIPTION
			FRWB, depending upon whether a double-precision array is BCD or binary.
FVIO	Establishes identification between a variable logical unit and the corresponding FORTRAN file.	.FVIO.	Entry point called for any FORTRAN input/output source statement that specifies a variable unit.
FRCD	Controls reading of cards on line and conversion of alphameric card code to BCD.	.FRCD.	Entry point for the source program statement READ format, list.
		.CARD.	Entry point called by subroutine FRWD for the reading of an input record when the input device is the on-line card reader.
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.

Subroutines .LXCON, .FPTRP, and FXEM must precede subroutine FOUT in the Subroutine Library. If subroutine FOUT is called, it overlays the calls to entry point .LXSEL, in these three subroutines, with calls to itself.

In addition, the FORTRAN input/output library contains the routines that produce the $FILE cards needed by the Loader for the initialization of IOCS files used by the object program.

Eight library routines, in the following form, are provided:

```
         ENTRY  .UNxx.
.UNxx.   PZE    .UNITn
UNITn    FILE   specifications
```

where n is a two-digit FORTRAN logical 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 an additional entry point, .BUFSZ, which is in the following form:

```
.BUFSZ    PZE    BUFSIZ
BUFSIZ    EQU    22
```

The variable field of BUFSIZ contains the maximum BCD logical record size. The loading of subroutine UN06 forces subroutine FRWD to use subroutine FOUT when the System Output Unit is to be written. Subroutine FRWD must precede subroutine UN06 in the Subroutine Library, so that the call to subroutine FOUT is inserted in subroutine FRWD.

The assembly of a library routine of this form produces a $FILE card with the given specifications, thereby enabling the Loader to establish correspondence between the FORTRAN logical unit n and the system file associated with it. Symbolic location .UNXX. is entered into the control dictionary as an external symbol by the ENTRY operation. File UNITn is entered into the file dictionary for this library subroutine. By virtue of the FILE operation code, whenever UNITn occurs in the variable field of any instruction, the relocatable reference is to the file dictionary entry for file UNITn. The $FILE card makes the correspondence between file name UNITn and the related system unit, as specified in the variable field of the FILE operation code. Thus, at execution time, the address field of symbolic location .UNXX. is set by the Loader to be the absolute address of the file control block of the corresponding unit.

CORRESPONDENCE BETWEEN FORTRAN LOGICAL UNITS AND SYSTEM FILES

Input/output devices are always referred to symbolically in FORTRAN input/output statements. Object program input/output operates through the FORTRAN IOCS buffering package. The correspondence between the symbolic unit reference and the actual physical unit is established in the initialization of IOCS by the Loader. As shown in the following chart, the standard FORTRAN input/output configuration allows for symbolic units 1 through 8. The normal unit designation for BCD input statements is unit 5; for BCD output statements, it is unit 6.

FORTRAN LOGICAL UNIT	SYSTEM FILE	MODE	FUNCTION
01	SYSUT1	Binary	Input or output
02	SYSUT2	Binary	Input or output
03	SYSUT3	Binary	Input or output
04	SYSUT4	Binary	Input or output
05	SYSIN1	BCD	Input
06	SYSOU1	BCD	Output
07	SYSPP1	Binary	Output
08	System Availability Chain	BCD	Input or output

The symbolic unit references may be changed by the installation in accordance with its own needs.

The actual specifications for the file corresponding to each symbolic unit are made through the $FILE cards in the FORTRAN Input/Output Library. However, an installation may change the file specifications for any given unit(s) by reassembling the library routine, which generates the $FILE card for the corresponding unit(s), with whatever options may be desired in the variable field of the FILE pseudo-operation. The publication *IBM 7090/7094 Programming Systems: Macro Assembly Program (MAP) Language,* Form C28-6311, contains further information on the FILE pseudo-operation. The standard FORTRAN file specifications have been listed previously. Since the density is not specified, high-density is assumed. If a system unit is assigned to a file, the file specifications for the system unit function override any density and file-closing specifications set by the generated $FILE card.

SUBROUTINE LIBRARY LISTING OUTPUT

The 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 is on, the output is in BCD mode, blocked up to five lines per block. When bit 2 is off, 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 binary 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, symbolic card 3F5UN650 must contain the following:

```
UNIT06    FILE    , OU, READY, OUTPUT, BLK = 110,
                      MULTIREEL, BCD, NOLIST
```

When the IBM 720 is used for output, the following changes must also be made:

1. Card 3F318850 in the FRWD subroutine must contain the following:

```
LIMIT   EQU     20
```

2. Card 3F5UN652 in the UN06 subroutine must contain the following:

```
BUFSIZ   EQU    20
```

If the System Output Unit (SYSOU1) is to be printed using the IBM 1401 Peripheral Input/Output Program, symbolic card 3F5UN650 must contain the following:

```
UNIT06    FILE    , OU, READY, OUTPUT, BLK = 116,
                      MULTIREEL, BIN, NOLIST
```

FORTRAN Utility Library

Some of the subroutines in the FORTRAN Utility Library are described in the following text. Other subroutines in this library are described in the section "The Subroutine Library (IBLIB)."

SUB-ROUTINE	DESCRIPTION	ENTRY POINT	DESCRIPTION
ERAS,	Erasable words used by object program.	E.1, E.2, E.3, E.4	Erasable words used by object program.
FPARST	Used by SIFT 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	Argument for the call to subroutine FOUT.
XIT	Returns control to subroutine .LXCON	EXIT	Entry point for termination of object program execution.

Object program errors found by the Subroutine Library call subroutine FXEM, which controls object program error procedures. A message stating the error condition is written on the System Output Unit. Normally, execution is terminated and control is given to subroutine .LXCON. However, the execution of the mathematical subroutines and some input/output subroutines can be resumed by using optional exits. These optional exits are controlled by bits in location OPTWD1 and OPTWD2. Error conditions 1-35 are controlled by bits 1-35 of OPTWD1, and error conditions 36-57 are controlled by bits 1-22 of OPTWD2. (The error lists are shown in Figures 21 through 25.) To use an optional exit, the bit associated with the relevant error condition must be set to 1. This can be done by changing locations OPTWD1 and/or OPTWD2, which are in control section .OPTW. of subroutine FXEM. The following MAP subroutine sets each relevant bit to 1, thereby allowing the use of all permissable optional exits:

```
          ENTRY  .OPTW.
.OPTW     OCT    377777777777
          OCT    076222420000
          END
```

The exits set by this procedure 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's routine and a nonstandard error code argument is used, the error code is written on the standard 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 if, in a MAP routine, a call is made to an entry point within the same routine. This cannot occur in a routine written in FORTRAN or in a subroutine in the Subroutine Library.

Subroutine in Which Error is Encountered	Error Code	Error Condition	Optional Exit
FXP1	1	For I^J where $I = 0$, $J = 0$	Set $I^J = 0$
	2	For I^J where $I = 0$, $J < 0$	Set $I^J = 0$
FXP2	3	For B^J where $B = 0$, $J = 0$	Set $B^J = 0$
	4	For B^J where $B = 0$, $J < 0$	Set $B^J = 0$
FXP3	5	For B^C where $B < 0$, $C \neq 0$	Evaluate for $\|B\|$
	6	For B^C where $B = 0$, $C = 0$	Set $B = 0$
	7	For B^C where $B = 0$, $C < 0$	Set $B = 0$
FXPF	8	For e^B where $B > 88.028$	Set $e^B = B$
FLOG	9	For log B where $B = 0$	Set log $B = 0$
	10	For log B where $B < 0$	Evaluate for $\|B\|$
FATAN	11	For arctan (A,B) where $A = 0$, $B = 0$	Set angle = 0
FSCN	12	For sin(A) or cos(A) where $\|A\| \geq 2^{27}$	Set result = 0
FSQR	13	For $A^{1/2}$ where $a < 0$	Evaluate for $\|A\|$

Figure 21. Single-Precision Mathematical Subroutines

FORTRAN Library Error Messages

The following list gives the subroutine in which the error was encountered, the error code, the error message, and the optional exit.

FXP1	1	EXPONENTIATION ERROR 0**0	Set result = 0
FXP1	2	EXPONENTIATION ERROR 0**(–J)	Set result = 0
FXP2	3	EXPONENTIATION ERROR 0**0	Set result = 0
FXP2	4	EXPONENTIATION ERROR 0**(–J)	Set result = 0
FXP3	5	EXPONENTIATION ERROR (–B)**C	Evaluate for + B
FXP3	6	EXPONENTIATION ERROR 0**0	Set result = 0
FXP3	7	EXPONENTIATION ERROR 0**(–C)	Set result = 0
FXPF	8	EXP(B),BGRT THAN 88.028, NOT ALLOWED	Set result = argument
FLOG	9	LOG(0) NOT ALLOWED ALLOWED	Set result = 0
FLOG	10	LOG(–B) NOT ALLOWED	Evaluate for + B
FATN	11	ATAN2(0,0) NOT ALLOWED	Set result = 0
FSCN	12	SIN OR COS ARG GRT TH 2**27 NOT ALLOWED	Set result = 0
FDX1	14	EXPONENTIATION ERROR 0**0	Set result = 0
FDX1	15	EXPONENTIATION ERROR 0**(–J)	Set result = 0
FDX2	16	EXPONENTIATION ERROR (–B)**C	Evaluate for + B
FDX2	17	EXPONENTIATION ERROR 0**0	Set result = 0
FDX2	18	EXPONENTIATION ERROR 0**(–C)	Set result = 0
FDXP	19	EXP(B),B GRT TH 88.028, NOT ALLOWED	Set result = argument

All these optional exits are standard
in distributed version of system Through 30 only.

Subroutine in Which Error is Encountered	Error Code	Error Condition	Optional Exit
FDX1	14	For D^I where $D = 0$ and $I = 0$	Set result = 0
	15	$D = 0$ and $I < 0$	Set result = 0
FDX2	16	For $D_1^{D_2}$ where $D_1 < 0$, $D_2 \neq 0$	Evaluate for $\lvert D_1 \rvert$
	17	$D_1 = 0$, $D_2 = 0$	Set $D_1^{D_2} = 0$
	18	$D_1 = 0$, $D_2 < 0$	Set $D_1^{D_2} = 0$
FDXP	19	For e^D where $D > 88.028$	Set $e^D = D$
FDLG	20	For log D where $D = 0$	Set log D = 0
	21	For log D where $D < 0$	Evaluate for $\lvert D \rvert$
FDSQ	22	For $D^{1/2}$ where $D < 0$	Evaluate for $\lvert D \rvert$
FDSC	23	For sin(D) or cos(D) where $D \geq 2^{54}$	Set result = 0
FDAT	24	For arctan (D_1, D_2) where $D_1 = 0$, $D_2 = 0$	Set angle = 0

Figure 22. Double-Precision Mathematical Subroutines

Subroutine in Which Error is Encountered	Error Code	Error Condition	Optional Exit
FCAS	25	For Z_1/Z_2 where $Z_2 = 0$	Set result = 0
FDX1	14	For Z^I where $Z = 0$ and $I = 0$	Set result = 0
	15	$Z = 0$ and $I < 0$	Set result = 0
FCXP	26	For e^Z where $X > 88.928$	Set $e^Z = Z$
	27	For e^Z where $Y \geq 2^{27}$	Set $e^Z = Z$
FCLG	28	For log (Z) where $Z = 0$	Set log Z = 0
FCSC	29	For sin(Z) or cos(Z) where $X \geq 2^{27}$	Set result = 0
	30	For sin(Z) or cos(Z) where $Y > 88.028$	Set result = 0

Figure 23. Complex Mathematical Subroutines, Where Z=X+iy

FDLG	20	DLOG(0) NOT ALLOWED	Set result = 0
FDLG	21	DLOG(–B) NOT ALLOWED	Evaluate for + B
FDSQ	22	SQRT(–B) NOT ALLOWED	Evaluate for + B
FDSC	23	DSIN OR DCOS ARG GRT TH 2**54 NOT ALLOWED	Set result = 0
FDAT	24	DATAN2(0,0) NOT ALLOWED	Set result = 0
FCAS	25	COMPLEX Z/O NOT ALLOWED	Set result = 0
FCXP	26	EXP(Z),REAL PART GRT88.028, NOT ALLOWED	Set result = argument
FCXP	27	EXP(Z),IMAG PART GRT 2**27, NOT ALLOWED	Set result = argument
FCLG	28	CLOG(0) NOT ALLOWED	Set result = 0
FCSC	29	CSIN OR CCOS ARG WITH REAL PART GRT 2**27 NOT ALLOWED	Set result = 0
FCSC	30	CSIN OR CCOS ARG WITH IMAG PART GRT 88.028 NOT ALLOWED	Set result = 0
FRWD	31	FORMAT AT XXXXX, FIRST WORD HAS ILLEGAL CONTROL CHARACTER OR SPECIFIES TOO LONG A LINE	Treat as end of format
FRWD	32	ILLEGAL CHAR IN DATA BELOW OR BAD FORMAT (RECORD CONTAINING ILLEGAL CHARACTER WRITTEN ON LINE FOLLOWING MESSAGE)	Treat illegal character as zero
FRWD	33	ILLEGAL CHAR IN DATA BELOW OR BAD FORMAT (RECORD CONTAINING ILLEGAL CHARACTER WRITTEN ON LINE FOLLOWING MESSAGE)	Treat illegal character as zero
FRWD	34	END OF FILE READING UNITXX	Read next file
FRWD	35	PERMANENT READ REDUNDANCY UNITXX	Record used as read the 100th time
FRWD	36	END-OF-BUFFER EXIT READING UNITXX	No optional exit-execution terminated

These optional exits are NON standard
To obtain the relevant [illegible]
must have been set to 1 cf p79.

Subroutine in Which Error is Encountered	Error Code	Error Condition	Optional Exit
FRWD	31	Invalid character in FORMAT statement.	Treat as end of FORMAT statement.
	32	Invalid characters in BCD input data.	Treat invalid character as zero.
	33	Invalid character in octal input data.	Treat invalid character as zero.
	34	End of file (not on System Input Unit).	Read next file.
	35	Permanent read redundancy.	Record used as read the 100th time.
	36	End-of-buffer exit reading.	No optional exit. Execution terminated.
	37	End-of-buffer exit writing.	No optional exit. Execution terminated.
	54	Write request on unit defined as System Input Unit.	No optional exit. Execution terminated.
	56	Read request on unit defined as System Output Unit.	No optional exit. Execution terminated.
	57	Invalid character in logical input data.	Treat invalid character as blank.
FRWB	38	Physical record size exceeds buffer size.	Process portion of record in buffer.
FRWB (cont'd)	39	Internal label word count does not agree with IOCS word count.	Process record read.
	40	List exceeds logical record length.	Store zeros in remaining list items.
	41	End of file reading.	Read next file.
	42	Permanent read redundancy.	Record used as read the 100th time.
	43	End-of-buffer exit reading.	No optional exit. Execution terminated.
	44	End-of-buffer exit writing.	No optional exit. Execution terminated.
FRCD	45	Invalid card character.	Ignore card. Read next card.
	46	End-of-file card reader.	No optional exit. Execution terminated.
FVIO	47	Logical unit not defined.	No optional exit. Execution terminated.
FBST	48	Permanent read redundancy.	Record used as read the 100th time.
	49	End-of-buffer exit reading.	No optional exit. Execution terminated.

Figure 24. Error Conditions Recognized by FORTRAN Input/Output Library

Subroutine in Which Error is Encountered	Error Code	Error Condition	Optional Exit
FDMP	50	Tape redundancy on System Utility File (SYSUT4) when attempting to write.	No optional exit. Execution terminated.
FSLITE	51	For I larger than 4 when setting the sense light.	No action is taken.
		For I equal to 0 or larger than 4 when testing the sense light.	Set J equal to 2 (OFF).
FSSWTH	53	For I larger than 6.	Set J equal to 2 (OFF).

Figure 25. Error Conditions Recognized by the System Monitor

FRWD	37	END-OF-BUFFER EXIT WRITING UNITXX	No optional exit — execution terminated
FRWB	38	PHYSICAL RECORD SIZE EXCEEDS BUFFER SIZE	Process portion of record in buffer
FRWB	39	INTERNAL LABEL WORD COUNT DOES NOT MATCH IOCS WORD COUNT	Process record read
FRWB	40	LIST EXCEEDS LOGICAL RECORD LENGTH	Store zeros in remaining list items
FRWB	41	END OF FILE READING UNITXX	Read next file
FRWB	42	PERMANENT READ REDUNDANCY UNITXX	Record used as read the 100th time
FRWB	43	END-OF-BUFFER EXIT READING UNITXX	No optional exit — execution terminated

FRWB	44	END-OF-BUFFER EXIT WRITING UNITXX	No optional exit — execution terminated
FRCD	45	ILLEGAL CARD CHARACTER	Ignore card. Read next card
FRCD	46	END-OF-FILE CARD READER	No optional exit — execution terminated
FVIO	47	LOGICAL UNIT NOT DEFINED FOR VALUE XX	No optional exit — execution terminated
FBST	48	PERMANENT READ REDUNDANCY UNITXX	Record used as read
FBST	49	END-OF-BUFFER EXIT READING UNITXX	No optional exit — execution terminated
FDMP	50	TAPE REDUNDANCY ON SYSUT4 ATTEMPTING TO WRITE MEMORY SAVE	No optional exit — execution terminated
FSLITE	51	REFERENCE TO NONEXISTENT SENSE LIGHT	Declared 'off' if testing Ignored if setting
FSSWTH	53	NONEXISTENT SENSE SWITCH TESTED	Switch declared 'up'
FRWD	54	WRITE REQUEST ON UNIT DEFINED AS SYSIN1 ILLEGAL	No optional exit — execution terminated
FRWD	55	ILLEGAL VALUE FOR COMPUTED GO TO AT IFN XXXXX	Execution terminated
FRWD	56	READ REQUEST ON UNIT DEFINED AS SYSOU1 ILLEGAL	No optional exit — execution terminated
FRWD	57	ILLEGAL CHAR FOR L CONVERSION IN DATA BELOW (RECORD CONTAINING ILLEGAL CHARACTER WRITTEN ON LINE FOLLOWING MESSAGE)	Treat illegal char as blank

In addition to the preceding messages, resulting from the recognition of an FXEM error condition, the following messages are also written by the Subroutine Library.

FPTRP	UNDRFLOW AT XXXXX IN AC UNDRFLOW AT XXXXX IN AC AN MQ UNDRFLOW AT XXXXX IN MQ OVERFLOW AT XXXXX IN AC OVERFLOW AT XXXXX IN AC AN MQ OVERFLOW AT XXXXX IN MQ ADDRESS AT XXXXX ODD

```
FXEM   ERROR TRACE. CALLS IN REVERSE ORDER.
       CALLING IFN OR ABSOLUTE
       ROUTINE LINE NO. LOCATION
       XXXXXX  XXXXX    XXXXX
       XXXXXX  XXXXX    XXXXX
         •       •        •
         •       •        •
         •       •        •
       XXXXXX  XXXXX    XXXXX
```

	ERROR MESSAGE OR ERROR CODE XXXXX NOT A STANDARD CODE DATA CONTAINING ILLEGAL CHARACTER IF PERTINENT EXECUTION TERMINATED OPTIONAL EXIT MESSAGE
FOUT	END-OF-BUFFER EXIT WRITING SYSOU1. EXECUTION TERMINATED
FBST	BACKSPACE REQUEST IGNORED ON UNITXX
FEFT	REQUEST TO WRITE EOF ON LIMIT ASSIGNED AS SYSIN1, SYSOU1, OR SYSPP1 HAS BEEN IGNORED
FRWT	REQUEST TO REWIND UNIT ASSIGNED AS SYSIN1, SYSOU1, OR SYSPP1 HAS BEEN IGNORED
FDMP	EXECUTION TERMINATED BY DUMP — DISK ERROR EXECUTION TERMINATED BY DUMP — UNUSUAL END SYSUT4 EXECUTION TERMINATED BY DUMP — EWA FLAG ON — SYSUT4 EXECUTION TERMINATED BY DUMP — LESS THAN 12 TRACKS ATTACHED TO SYSUT4
DMPR	PLEASE SUPPLY CORRECT CALLING SEQUENCE FOR DUMP EXECUTION TERMINATED BY DUMP — SYSUT4 REDUNDANCY EXECUTION TERMINATED BY DUMP — UNUSUAL END — SYSUTA EXECUTION TERMINATED BY DUMP — DISK ERROR EXECUTION TERMINATED BY DUMP — EWA FLAG ON — SYSUT4 EXECUTION TERMINATED BY DUMP — SYSOU1 REDUNDANCY. EXECUTION TERMINATED BY DUMP — UNUSUAL END — SYSOU1 SYSOU1 IS NOW XHK/S MOUNT NEW TAPE ON XHK/S

COBOL Object-Time Subroutines

This section contains a description of object time subroutines that are pertinent to COBOL. The COBOL Compiler generates instructions that call the appropriate object time subroutines from the System Library. The Loader completes the generated program by loading the desired subroutines and adjusts all instructions that refer to the subroutines so as to reflect the assigned subroutine locations.

The following types of subroutines are contained in this section:

1. MOVPAK subroutines — A group of COBOL subroutines concerned with the movement, conversion, and editing of data.
2. Additional COBOL subroutines — A group of arithmetic, conversion, and comparison subroutines.
3. IOCS subroutines — A group of pertinent IOCS subroutines.

MOVPAK Subroutine Special Locations

The following special locations are used by MOVPAK.

```
. CAREF    PZE    location, , byte
```

This location indicates the first-word address and the first-byte position of the source field involved in a move. A byte is defined as a group of consecutive binary digits. The first-byte position consists of digits 0 to 5. The preceding location is set in one of the following ways:

1. It is set automatically by calls to MOVPAK entry points .CMPAK and .CMPK2. This is described in the section "MOVPAK Major Entry Points," in items 1 and 3.
2. It is set by in-line coding preceding calls to .CMPK1 or .CMPK3. This is described in the section "MOVPAK Major Entry Points," in items 2 and 4. The in-line coding which sets the location is in one of the following three forms:
   a. The following coding is used when the data item is in working storage. SP+nnn contains the location and the byte of the data item.

      ```
      LDI    SP+nnn
      STI    . CAREF
      ```

   b. The following coding is used when the data item is located by means of a simple base locator. A simple base locator is defined as one that always has a byte equal to zero.

      ```
      CAL    BL+nnn
      TZE    . CBLER
      ACL    SP+nnn
      SLW    . CAREF
      ```

      SP+nnn is a constant of the following form:

      ```
      PZE    displacement, , byte
      ```

      .CBLER is the name of an error subroutine to which control is transferred if the base locator has not been properly set.
   c. The following coding is used when the data item is located by means of a complex base locator. A complex base locator may have a byte value other than zero.

      ```
      CAL    BL+nnn
      TZE    . CBLER
      ACL    SP+nnn displacement constant
      PDX    0,4
      TXL    *+2,4,5
      ACL    SP+mmm OCT 777772000000
      SLW    . CAREF
      ```

```
. CBREF    PZE    location , , byte
```

This location indicates the first word address and the first byte of the target field involved in a move. The location is set by calls to MOVPAK entry points. .CMPAK or .CMPK1 or by in-line coding of the same general form as that described for the setting of .CAREF.

```
. COFLO    PZE    **
```

This location is set to nonzero whenever any one of the numeric move or convert MOVPAK subroutines detects the truncation of significant high-order digits. The location is tested to determine whether a SIZE ERROR has occurred.

```
. CUFLO    PZE    **
```

This location is set to nonzero whenever a floating-point underflow results from a move. At present, no generated instructions test the status of the location.

MOVPAK Major Entry Points

The following discussion describes the four major entry points to MOVPAK subroutines:

```
TSX          . CMPAK, 4
             source address reference
             target address reference.
             (begin specific move call)
```

This entry uses the source and target address reference information to set the contents of the .CAREF and .CBREF locations. Control is then transferred to the subsequent string of instructions to perform the move. The address reference word is in one of the following forms:

1. The following coding is used when the data item is in working storage.

   ```
   PZE    location,, byte
   ```

2. The following coding is used when the data item is located by a base locator.

   ```
   MZE    BL+nnn,,SP+nnn
   ```

   BL+nnn gives the base locator reference and SP+nnn contains the item's displacement from the base locator (with the word displacement in the address of the constant and the byte displacement, if any, in the decrement).
3. The following coding is used when the data item is located by a positional indicator.

   ```
   MON    PI+nnn
   ```

   PI+nnn is the reference to the particular position indicator.

```
TSX    . CMPK1,4
target address reference
(begin specific move call)
```

This is the entry to MOVPAK that is used when no source address is necessary or when the source field is in an arithmetic register.

```
TSX    . CMPK2,4
source address reference
(begin specific move call)
```

This is an entry to MOVPAK that is used when the resultant field is to be left in an arithmetic register.

```
TSX    . CMPK3,4
(begin specific move call)
```

This is an entry to MOVPAK that is used when the source field is in an arithmetic register and the result is to be left in an arithmetic register or when any necessary field address references have been previously stored in locations .CAREF and/or .CBREF. This entry is also used when any necessary address references have previously been stored by in-line instructions into locations .CAREF and/or .CBREF.

General Form of MOVPAK Subroutine Calls

Calls upon specific subroutines within MOVPAK begin with a TXI instruction which transfers control to the entry point of the particular subroutine. The TXI instruction appears after a TSX instruction which has transferred control to one of the four major MOVPAK entry points. Some of the calls are fixed length; other calls are terminated by a TXI instruction which transfers control to a particular location.

Field Types

The following abbreviations are used in the discussions of the types of fields involved in move operations:

AA	Alphabetic field (The PICTURE clause contains only As.)
AN	Alphameric field (The group item or PICTURE clause contains Xs.)
RP	Report field (The PICTURE clause contains editing characters.)
XD	External decimal (fixed-point BCD)
ID	Internal decimal (fixed-point binary, scaled decimal)
IN	Internal decimal not SYNCHRONIZED RIGHT
SD	Scientific decimal (floating-point BCD)
FP	Floating point (floating binary)

The following are figurative constants:

SP	SPACES
ZE	ZEROS
CH	Characters (this category includes all one-character-literal, QUOTE, and HIGH-VALUE constants)

Literals are classed as AN, ID, or FP, as appropriate.

MOVPAK SUBROUTINE CALLS

The following calls are given in alphabetical order by the abbreviated representations for the source and target fields. The order is AA, AN, CH, FP, ID, IN, RP, SD, SP, XD, and ZE. For example, the move from internal decimal to external decimal is designated by the letters IDXD. Combinations other than those shown are not permitted.

AAAA, AAAN, ANAA, ANAN

Moves of most simple BCD, AN, or AA fields are handled by generated in-line instructions if the fields are short enough, but other cases are handled by one of the following calls:

Noncomplex move

```
TXI     .CANA1, 1, number. of. characters. to.
        move
```

Noncomplex move and trailing spaces

```
TXI     .CANA2, 1, number. of. characters. to.
        move
```

```
TXI     .CANA3, 1, number. of. spaces. to.
        insert
```

Complex Move – The length and/or initial byte position of the source field and/or the target field was not known at compilation time.

```
TXI     .CANA4, 1, control. 0
PZE     control. 1, , control. 2
```

1. *control. 1* is the length of the source field or the location of the word containing that information.

2. *control. 2* is the length of the target field or the location of the word containing that information.

The precise nature of control. 1 and control. 2 is defined by control. 0 as follows:

CONTROL. 0 VALUE	EXPLANATION
1	Target field length is in words (not characters).
2	Source field length is in words (not characters).
4	Control. 2 is the target length location (not the length itself).
8	Control. 1 is the source length location (not the length itself).

The preceding conditions may exist in combination, giving control. 0 a maximum value of 15.

ANFP, ANID, ANIN, ANRP, ANSD, and ANXD

Substitute XD for AN; then see the equivalent section.

.CEXAM

```
TSX     .CMPK2, 4
data-item reference
TXI     entpt, 1, length of data item parameter
        word
```

This subroutine is called by .CMPK2, and it processes text strings created by the EXAMINE and IF (CLASS) analyzers. The data-item reference word conforms to normal MOVPAK specifications. The entpt is .CEXAM for true EXAMINE statements;

.CXAMA for IF . . . ALPHABETIC; and
.CXAMN for IF . . . NUMERIC.

CHAN

```
TXI     .CCHAN, 1, number. of. characters. to.
        insert
OCT     characters
```

The second word contains six characters of the type to be inserted.

FPAN

Substitute XD for AN; then see the equivalent section.

FPFP

In-line instructions are used. MOVPAK is not involved.

FPID

```
TSX     CF1ID, 4
target. control. word. type. ID
```

This MOVPAK subroutine converts the floating-point value in the accumulator to internal decimal and leaves the result in the accumulator or the AC-MQ. This is a direct entry to MOVPAK and is not preceded by the normal TSX instruction to one of the four major entry points. An alternate form of the call (TSX ☐☐.CF2ID, 4) is used if the floating-point number is double precision and appears in the AC-MQ.

Target. control. word. type. ID contains the following information:

1. Prefix Portion – The sign of the scale is plus if the prefix is PZE and minus if the prefix is MZE.

2. Address Portion – The scale applied in the PICTURE clause of the internal decimal item.

3. Decrement Portion – The number of nines in the PICTURE clause of the internal decimal item. If the number is greater than ten, double-precision treatment is required.

FPIN

See FPID; then see IDIN.

FPRP

See –FPID; then see IDRP (omitting the IDID step).

FPSD

```
        TXI     .CF1SD, 1, 0
        target. control. word. type. SD
```

This MOVPAK subroutine converts the floating-point value in the accumulator to scientific decimal. An alternate form of the call (TXI .CF2SD, 1, 0) is used if the floating-point number is double precision and appears in the AC-MQ.

Target. control. word. type. SD contains the following information:

1. Prefix Portion – MZE, if a decimal point appears in the PICTURE clause; otherwise, it is PZE.

2. Address Portion – Scale, applied to the mantissa in the PICTURE clause.

3. Tag Portion – The sign convention code based on the PICTURE clause.

0	mantissa and exponent sign conventions are –.
1	mantissa – and exponent +.
2	mantissa + and exponent –.
3	mantissa + and exponent +.

4. Decrement Portion – The total length of the field, in characters.

FPXD

See FPID; then see IDXD.

IDAN

Substitute XD for AN; then see the equivalent section.

IDFP

```
        TSX     .CIDF1, 4
        source. control. word. type. ID
```

This MOVPAK subroutine converts the internal decimal value in the accumulator or AC-MQ to floating-point and leaves the result in the accumulator. Note that this is a direct entry to MOVPAK and is not preceded by the normal TSX instruction to one of the four major entry points. An alternate form of the call (TSX .CIDF2, 4) is used to develop double-precision floating-point results in the AC-MQ.

See FPID for the form of source. control. word. type. ID.

IDID

MOVPAK is not used. In-line scaling instructions are generated as appropriate, unless IDID is being used as an intermediate stage of a multistage move, where the scaling function is performed by a MOVPAK subroutine; e.g., see XDSD.

IDIN (See IDID first.)

```
        TXI     .CIDIN, 1, character. length. of .target
```

The source field is in the accumulator or in the AC-MQ. Character. length. of. target is the least multiple of six bits that is sufficient to contain the defined internal decimal field and its sign.

IDRP (See IDID first.)

```
        TXI     .CIDRP, 1, number. of. digits. to. con-
                vert
```

This instruction is followed by one or more instructions from the Report Field TXI instruction set. The particular instructions used reflect the characters that form the PICTURE of the field.

The members of the set are as follows:

```
        TXI     .CR999, 1, number. of. consecutive. 9.
                                            occurrences
        TXI     .CRZZZ, 1, number. of. consecutive. Z.
                                            occurrences
        TXI     .CRAAA, 1, number. of. consecutive. *.
                                            occurrences
        TXI     .CR000, 1, number. of. consecutive. 0.
                                            occurrences
        TXI     .CRBBB, 1, number. of. consecutive. B.
                                            occurrences
        TXI     .CRSIN, 1, C1+64*C2
```

The character C1 is inserted if the sign of the field is plus; the character C2 is inserted if the sign of the field is minus.

```
        TXI     .CRSIG, 1, C3+64*C4
```

The character C4 is inserted if no preceding significant digit has been inserted; the character C3 is inserted if a preceding significant digit has been inserted. If a (TXI CRFLS) instruction (described next) has been executed and the floating sign has not yet been inserted, the character actually inserted is C4, C5, or C6.

```
        TXI     .CRFLS, 1, C5+64*C6
```

C5 is the floating-sign character that is ultimately inserted if the sign of the field is plus, and C6 is ultimately inserted if the sign of the field is minus. If the first digit value is zero, a blank or the appropriate choice of C5 or C6 is immediately inserted as a result of this TXI instruction. Note that the first floating-sign position is traversed by this TXI instruction.

The following coding is used when other floating-sign positions are to follow.

```
          TXI     .CRFFF, 1, number. of. consecutive.
                             floating. sign. occurrences
```

The following coding is used when no other floating-sign positions are to follow.

```
          TXI     .CRFFQ, 1, number. of. consecutive.
                             floating. sign. occurrences
```

The following coding is used when no other floating-sign occurrences are to follow but there is a comma before the next digit.

```
          TXI     .CRFFC, 1, number. of. consecutive.
                             floating. sign. occurrences
```

Values which C1-C2, C3-C4, and C5-C6 can assume as character pairs are as follows:

C1	+	space	space	space	space	space	.$
C2	–	–	C	R	D	B	.$
C3	,	,	,				
C4	,	space	*				
C5	+	space	$				
C6	–	–	$				

The report image TXI string is terminated by the following:

```
          TXI     .CRQT, 1, value
```

where *value*=0 unless the field is to be blank when zero, in which case *value* is equal to the total length of the target field.

Example 1: PICTURE IS $$$, $$$. 99 is handled by the following:

```
          TXI     .CRFLS, 1, 43+64*43      C5=C6=$
          TXI     .CRFFF, 1, 2
          TXI     .CRSIG, 1, 59+64*48      C3=, and
                                           C4=space
          TXI     .CRFFQ, 1, 3
          TXI     .CRSIN, 1 ,27+64*27      C1=C2=.
          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      C3=, and
                                           C4=space
          TXI     .CRZZZ,1, 3
          TXI     .CRSIN, 1, 27+64*27      C1=C2=.
          TXI     .CR999, 1, 2             note choice
                                           of. CR999
          TXI     .CRSIN, 1, 16+64*32      C1=+ and
                                           C2=–
          TXI     .CRQIT, 1, 11            note blank
                                           when zero
                                           option
```

IDSD

```
          TXI     .CIDSD, 1, 0
          source. control. word. type. ID
          target. control. word. type. SD
```

The internal decimal contents of the accumulator or the AC-MQ are converted to scientific decimal form.

Source. control. word. type. ID:
(See FPID for the form.)

Target. control. word. type. SD:
(See FPSD for the form.)

IDXD (See IDID first.)

The internal decimal contents of the accumulator or the AC-MQ are converted to external decimal by one of the following three calls.

The following coding is used when the target field has no sign provision.

```
          TXI     .CIDX1, 1, number. of. characters. to.
                                                develop
```

The following coding is used when the target field has a sign over the low-order digit.

```
          TXI     .CIDX2, 1, number. of. characters. to.
                                                develop
```

The following coding is used when the target field always has a sign over the low-order digit.

```
          TXI     .CIDX3, 1, number. of. characters. to.
                                                develop
```

INAN
See INID; then see IDAN.

INFP
See INID; then see IDFP.

INID (See IDID afterwards.)

```
          TXI     .CINID, 1, character. length. of. source
```

The results are left in the accumulator or in the AC-MQ.

Character. length. of. source is the least multiple of six bits that is sufficient to contain the defined internal decimal field and its sign.

ININ
See INID; then see IDIN.

INRP
See INID; then see IDRP.

INSD
See INID; then see IDSD.

INXD
See INID; then see IDXD.

RPAN
See ANAN.

SDAN
Substitute XD for AN; then see the equivalent section.

SDFP

```
          TXI     .CSDF1, 1, 0
          source. control. word. type. SD
```

This subroutine converts the free-form contents of the scientific decimal field to single-precision floating-point in the accumulator. An alternate form of the call (TXI .CSDF2, 1, 0) is used to develop double-precision results in the AC-MQ.

Source. control. word. type. SD:
(See FPSD for the form.)

SDID

```
          TXI     .CSDID, 1, 0
          source. control. word. type. SD
          target. control. word. type. ID
```

This subroutine converts the free-form contents of the scientific decimal field to internal decimal and leaves the results in the accumulator or in the AC-MQ.

Source. control. word. type. SD:

(See FPSD for the form.)

Target. control. word. type. ID:

(See FPID for the form.)

SDIN

See SDID; then see IDIN.

SDRP

See SDID; then see IDRP.

SDSD

```
        TXI    .CSDSD, 1, 0
               source. control. word. type. SD
               target. control. word. type. SD
```

This subroutine converts the free-form contents of the source scientific decimal field to the form dictated by the target scientific decimal field.

Source. control. word. type. SD:

Target. control. word. type. SD:

(See FPSD for the form.)

SDXD

See SDID; then see IDXD.

SPAA, SPAN, SPRP, SPSD, SPXD

```
        TXI    .CSPAN, 1, number. of. spaces. to. insert
```

XDAN

See ANAN.

XDFP

See XDID; then see IDFP.

XDID (See IDID afterwards.)

```
        TXI    .CXDID, 1, number. of. characters. to. convert
```

This subroutine converts data from external decimal to internal decimal data and leaves the results in the accumulator or in the AC-MQ. The sign of the source field is assumed to be over the low-order digit. The absence of a sign is treated as denoting plus. Leading spaces appearing in the source field are treated as zeros.

XDIN

See XDID; then see IDIN.

XDRP

```
        TXI    .CXDRP, 1, 0
```

This instruction is followed by one or more instructions from the external decimal TXI instruction set (see XDXD for a definition of this set). The next instruction thereafter is of the following form:

```
        TXI    .CXDRQ, 1, number. or. digits. developed. for. target
```

This instruction is then followed by one or more instructions from the report field TXI instruction set (see IDRP for a definition of this set). As with IDRP, the calling sequence is terminated by the following:

```
        TXI    .CRQIT, 1, value
```

where value=0 unless the field is to be blank when zero, in which case value is equal to the total length of the target field.

XDSD

See XDID; then see IDSD.

XDXD

```
        TXI    .CXDXD, 1, target. sign. convention
```

Target. sign. convention is zero if the target field has no sign provision; it is one if the target field has a sign over the low-order digit when minus, and is two if the target field always has a sign over the low-order digit.

This instruction is followed by one or more instructions from the external decimal TXI instruction set. The particular instructions used represent a procedural method of construction of the proper string of digits for the target field. The members of the set are:

```
        TXI    .CXMOV, 1, number. of. digits. to. move
        TXI    .CXNZT, 1, number. of. digits. to. test. for. nonzero
        TXI    .CXBYP, 1, number. of. digits. to. bypass
        TXI    .CXINZ, 1, number. of. digits. to. insert
        TXI    .CXRND, 1, 0
```

If nonzero significance is encountered under control of .CXNZT, it causes .COFLO to be set nonzero. .CXRND is used when rounding is desired at the current position.

Three alternate subroutine entry points .CXMVS, .CXNZS, and .CXBYS) corresponding to .CXMOV, .CXNZT, and .CXBYP are used instead when there may be a sign over the last digit traversed under control of the instruction.

The instruction string is terminated for XDXD moves by the following:

```
        TXI    .CXDXQ, 1, number. of. digits. developed. for. target
```

Example 1: The statement COMPUTE A ROUNDED = B, ON SIZE ERROR . . . (where AS PICTURE IS S999V999 and BS PICTURE IS S9V9) results in the following:

```
        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 AS PICTURE IS S9V99 and BS PICTURE IS V9999) results in the following:

```
        TXI    .CXDXD, 1, 0
        TXI    .CXBYP, 1, 1
        TXI    .CXMVS, 1, 2
        TXI    .CXINZ, 1, 2
        TXI    .CXDXQ, 1, 4
```

ZEAN

```
        TXI    .CZEAN, 1, number. of. zeros. to. insert
```

ZEFP, ZEID
MOVPAK is not used. The value zero is stored by one or more in-line instructions.

ZEIN
See ZEAN.

ZERP
A sufficient number of zero digits is provided by generated in-line instructions which insert zeros in one or more temporary storage words. The move is then performed as if it were XDRP.

ZESD
The accumulator is cleared to zero by a generated in-line instruction. The move is then performed as if it were FPSD.

ZEXD
See ZEAN.

Additional COBOL Subroutines

In the discussions that follow, the symbol for the subroutine or the communication location is given at the left. The calling sequence of the subroutine or the communication location is indicated. An explanation then follows this coding.

.CARS1

.CARS2

These two locations serve as storage for multiprecision arithmetic operations.

.CAR01
```
        TSX     .CAR01, 4
        PZE     CP+nnn
```

This subroutine raises the double-precision floating-point number in the AC-MQ to the single-precision power stored in .CARS1. The double-precision result is left in the AC-MQ.

CP+nnn contains a constant which is the source language card number at which the original computation was specified. It is used only for an error message.

.CAR02
```
        TSX     .CAR02, 4
        PZE     CP+nnn
```

This routine raises the double-precision floating-point number in the AC-MQ to the double-precision power stored in .CARS1 and .CARS2. The double-precision result is left in the AC-MQ.

CP+nnn is the same as for .CAR01.

.CAR03
```
        TSX     .CAR03, 4
```

This is the subroutine for sign adjustment of double-precision, fixed-point numbers. The routine is entered with the number in the AC-MQ, and the result is left in the AC-MQ.

.CAR04
```
        TSX     .CAR04, 4
        PZE     CP+nnn
```

This routine scales up the single-precision number in the accumulator by 10**10 and then scales it up by the constant located at CP+nnn.

.CAR05
```
        TSX     .CAR05, 4
        PZE     CP+nnn
```

This routine scales up the number in the MQ by 10**10 and then scales it up by the constant located at CP+nnn.

.CAR06
```
        TSX     .CAR06, 4
        PZE     CP+nnn
```

This routine scales up the double-precision number in the AC-MQ by the constant located at CP+nnn.

.CAR07
```
        TSX     .CAR07, 4
        PZE     CP+nnn
```

This routine scales up the double-precision number in the AC-MQ by the constant located at CP+nnn. On entry to the routine, the high-order part of the number is in the MQ and the low-order part of the number is in the accumulator.

.CAR08
```
        TSX     .CAR08, 4
        PZE     CP+nnn
```

This routine scales down the double-precision number in the AC-MQ by the constant located at CP+nnn and leaves the result in the AC-MQ.

.CAR09
```
        TSX     .CAR09, 4
        PZE     CP+nnn
```

This routine scales down the double-precision number in the AC-MQ 10**10 and then scales it down by the constant located at CP+nnn, leaving the result in the MQ.

.CAR10
```
        TSX     .CAR10, 4
```

This routine divides the double-precision, fixed-point number in .CARS1 and .CARS2 by the contents of the AC-MQ. The result is left in the AC-MQ.

.CAR11
```
        TSX     .CAR11, 4
        PZE     CP+nnn
```

This routine multiplies the double-precision contents of the AC-MQ by the double-precision value in .CARS1 and .CARS2, scales down the product by the constant located at CP+nnn, and leaves the result in the AC-MQ.

.CAR12
```
        TSX     .CAR12, 4
        PZE     CP+nnn
```

This routine multiplies the double-precision contents of the AC-MQ by the double-precision value in .CARS1 and .CARS2, scales the product down by 10**10, and then scales it down by the constant located at CP+nnn. The result is left in the AC-MQ.

.CAR13

```
        TSX     .CAR13, 4
        PZE     CP+nnn
```

This floating-point exponential routine raises the single-precision number in the accumulator to the single-precision number located in .CARS1. The number in the accumulator may be a positive real number or a negative integer. The exponent may have any value. The single-precision result is left in the accumulator. .CAR13 calls .CEXPR and, in the event of an error, calls .CEXNG.

CP+nnn is the source language card number.

.CAR14

```
        TSX     .CAR14, 4
        PZE     CP+nnn
```

This floating-point exponential routine raises the single-precision number in the accumulator to the double-precision power located in .CARS1 and .CARS2. The number in the accumulator may be a positive real number or a negative integer. The exponent may have any value. The double-precision result is left in the AC-MQ. The deck .CAR14 contains the entry points .CAR14, .CAR01, and .CAR02. A call is made to .CEXPR and, in the event of an error, to .CEXNG.

.CBDCV

```
        TSX     .CBDCV, 4
        (Return)
```

This subroutine converts the BCD control word that precedes each variable-length record to binary form. Upon entry to this subroutine, the control word is in MQ. The word count of the record is contained in the first five characters, and the sixth character is a peripheral code. After conversion, the actual record length is in binary form in the decrement of the AC.

.CBNCV

```
        TSX     .CBNCV, 4
        (Return)
```

This subroutine converts the binary record length to BCD form and adds a word at the beginning of every variable-length record. Upon entry to this subroutine, the record length in number of words is in the MQ. The length is converted to BCD and placed in the first five characters of the control word. The sixth character of the control word is zero. After conversion, the control word is in the logical accumulator.

.CCTAB

This is a conversion table used in converting from the 7090 collating sequence to the COLLATE-COMMERCIAL collating sequence.

.CCOMP

```
        TSX     .CCOMP, 4
        OP      .CCTAB, , 6
        PZE     LOC(1), T(1), LOCATOR(1)
        PZE     LENGTH(1), , 6*BYTE(1)
        PZE     LOC(2), T(2), LOCATOR(2)
        PZE     LENGTH(2), , 6*BYTE(2)
                HIGH RETURN from comparison
                EQUAL RETURN from comparison
                LOW RETURN from comparison
```

This subroutine performs an alphabetic comparison on two fields.

OP is a CVR or a NOP, depending on the need to adjust the collating sequence before the comparison.

LOC(N) is the displacement from the base, if there is a base. If there is no base, it is the location of the field.

LOCATOR(N) is the location of the base locator. It is zero if there is no base locator, in which case T(N) is meaningless and must also be zero.

T(N) is nonzero if the base locator is complex.

LENGTH(N) is the length of the field in characters.

BYTE(N) is the nominal byte position.

.CHBCD

```
        TSX     .CHBCD, 4
        PZE     BL+nnn, , TS+8
        (Return)
```

This subroutine converts a card image to a BCD image. The card image is located in an area located by the base locator, and the BCD image is placed into the twelve-word temporary storage area beginning at TS+8.

.CBCDH

```
        TSX     .CBCDH, 4
        PZE     BL+nnn, , TS+8
        (Return)
```

This subroutine converts a BCD image to a card image.

The BCD image is located in the twelve-word temporary storage area beginning at TS + 8, and the card image is placed into the area located by the base locator.

Input/Output Subroutines

.CIOHS

```
        PZE     0, 0, **
```

This is a communication location which contains in its decrement the source language card number of the most recent input/output action. This location is set by every OPEN, CLOSE, READ, and WRITE in the source program and is therefore an input/output history location.

.CEOBP

This subroutine is associated with an IOBS .READ or .WRITE calling sequence. Control is transferred to .CEOBP when the IOBS end-of-buffer error condition is encountered during a .READ or .WRITE calling sequence. The subroutine displays an error message and causes object-time processing to terminate. The following is a typical calling sequence which contains a reference to .CEOBP:

```
        TSX     .WRITE, 4
        PZE     file-name, , . CEOBP
        IOSTN*  BL+nnn, , integer
```

.CERRP

This subroutine is associated with an IOBS .READ calling sequence. Control is transferred to .CERRP when the IOBS error condition is encountered during a .READ calling sequence. This subroutine displays an error message and causes object-time processing to terminate. The following is a typical calling sequence which contains a reference to .CERRP:

```
          TSX      .READ, 4
          PZE      file-name, , .CEOBP
          PZE      ENnnnn, , .CERRP
          IOSTN*   BL+nnn, , integer
```

.CEXNG

```
          TSX      .CEXNG, 4
          PZE      CP+nnn
```

This error routine, entered only by .CAR01, .CAR02, .CAR13, and .CAR14, prints an off-line message indicating an out-of-range condition in the factors of an exponentiation. Subroutine .CDPLY is called. CP+nnn is the same as for .CEXPR.

.CEXPR

```
          TSX      .CEXPR, 4
          PZE      CP+nnn
```

This subroutine, entered only by .CAR01, .CAR02, .CAR13 and .CAR14, checks for out-of-range conditions and overflow. Deck .CEXPR contains the entry point .CEXNG. CP+nnn is the source card number.

This subroutine determines if a file is attached to card equipment.

.COPEN

```
          TSX      .COPEN, 4
          ROP      file-name, , OPT
```

This subroutine opens a file. ROP is the rewind option, according to the following convention:

PZE	Rewind
MZE	No Rewind
MON	No Rewind, No Label Action

OPT equals 1 if the file name refers to an optional file.

.CCLOS

```
          TSX      .CCLOS, 4
          ROP      file-name, , OPT
          (Return)
```

This subroutine closes a file. The following conventions are used with the rewind option (ROP).

PZE	Rewind and Unload
PTW	Rewind
MZE	No Rewind
MON	No Rewind, No End-of-File Mark
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.

. CDPLY

```
          TSX      . CDPLY, 4
          PZE      IMAGE, , DEVICE
          (Return)
```

This is a subroutine used to display a twelve-word BCD image. IMAGE is the location of a twelve-word BCD area which is to be displayed, and DEVICE is an actual bit configuration within which:

Bit 15 = 1	if the area is to be displayed upon the PRINTER.
Bit 14 = 1	if the area is to be displayed upon the SYSTEM-OUTPUT-UNIT.
Bit 13 = 1	if the area is to be displayed upon the CARD-PUNCH. This option is not currently implemented.

. CKEYS

```
          TSX      . CKEYS, 4
          (Return)
```

This subroutine places the panel key setting in the MQ register.

. CBLER

This subroutine prints an error message and causes processing to terminate whenever a reference is made to a base locator which has not been set. It is normally reached by the following:

```
          LAC      BL+nnn, N
          TXL      . CBLER, N, 0
```

. CCDTY

```
          TSX      . CCDTY, 4
          PZE      file-name
          (return if file is attached to card equipment)
          (return if file is not attached to card equipment)
```

The Librarian

The Librarian is a section of the Loader that maintains the Subroutine Library. It is composed largely of routines from sections 1 and 2 of the Loader. The Librarian is called by the Loader upon encountering a $EDIT card following the $IBJOB card. A specification on the $EDIT card allows the programmer to obtain a listing of pertinent information about the Subroutine Library.

Operation of the Librarian is governed by the following control cards:

```
1             16
$REPLACE      srname    [, ORG = nnnnn]
$INSERT       [srname]  [, ORG = nnnnn]
$AFTER        srname
$DELETE       srname
$ASSIGN       srname, ORG = nnnnn
```

The field represented as srname is the name of a subroutine existing in the Subroutine Library on which the indicated operation is to be performed. It is optional on the $INSERT card and mandatory on all the others.

The field represented as ORG = nnnnn is the absolute origin to be assigned to the subroutine. It is mandatory only on the $ASSIGN card.

The Subroutine Library consists of the following two files of information:

1. The Control Information File, consisting of the Subroutine Section Name Table (SRNT), the Subroutine Dependence Table (SRDT), 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 which have text.

The library maintenance operation is essentially a four-phase process. SRNT and SRDT are spaced over 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 to 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 is 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 specified on the $EDIT card, a listing is prepared showing the subroutine name, the origin, if fixed, the length, and the starting record number of each subroutine. A listing 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 performed to replace the existing library files by the two new library files generated by the Librarian on the System Utility Unit (SYSUT4). If the System Utility Unit (SYSUT4) is attached to disk, the maximum block size is 464 words. If SYSUT4 is attached to any other device, the maximum block size is 524 words.

The Librarian is called by the Load Supervisor upon encountering the following card immediately after the $IBJOB card.

```
1              16
$EDIT          [LOGIC]
```

The LOGIC option, if specified, instructs the Librarian to provide information showing the cross-referencing of subroutines in the Library.

Restrictions Using Drum Storage

If the Subroutine Library is to reside on disk storage, the System Utility Unit (SYSUT4) must be attached to disk to obtain the proper block size. If SYSUT4 is not attached to disk, SYSUT3 may not be attached to disk.

Librarian Control Cards

$REPLACE Card

The format of this control card is:

```
1              16
$REPLACE       srname[, ORG = nnnnn]
```

The $REPLACE card causes the Librarian to copy the current library up to, but not including, the subroutine named srname. The named subroutine is then skipped over in the current library and the subroutine deck following the $REPLACE card is inserted in the output library files. The name for the new subroutine is obtained from the $IBLDR card of the deck.

The optional field ORG=nnnnn is used to assign an absolute origin to the subroutine which is being inserted or to change its assembled absolute origin. The five-digit field nnnnn is the absolute origin, in octal.

$ASSIGN Card

The format of this control card is:

```
1              16
$ASSIGN        srname, ORG = nnnnn
```

The $ASSIGN card causes the Librarian to copy the current library up to, but not including, the subroutine name 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.

Another control card or an end of file must follow the $ASSIGN card.

$INSERT Card

The format of this control card is:

```
1              16
$INSERT        [srname][, ORG = nnnnn]
```

The $INSERT card causes the Librarian to place the subroutine deck that follows into the library at its current position. 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 being made at the current position of the output library file.

$AFTER Card

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 named 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 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 named srname. The named subroutine is then skipped over on the current library. The subroutine name is mandatory on this control card.

Another control card or an end of file must follow the $DELETE card.

Appendix A: Control Card Format Index

Refer to the given page reference for a description of each card.

CARD FORMAT 1	8	16	PAGE REFERENCE
$*		any text	9
$AFTER		srname	94
m	*ALTER	n	12
m	*ALTER	n1 ,n2	12
$ASSIGN		srname, ORG=nnnnn	93
$CBEND			16
$DELETE		srname	94
$DUMP[n]	cxxxxx	loc1/loc2, loc3/loc4, . . .	44

CARD FORMAT

1	8	16	PAGE REFERENCE
	*ENDAL		12
$ENTRY		$\left[\left\{\begin{matrix}\text{exname}\\\text{deckname}\end{matrix}\right\}\right]$	9
$ETC			25
$EXECUTE		subsystem name	7
$FILE		'filename' [,unit1, unit2] $\left[,\left\{\begin{matrix}\underline{\text{LIST}}\\\text{NOLIST}\end{matrix}\right\}\right]$	20
$ETC		$\left[,\left\{\begin{matrix}\underline{\text{MOUNT}}\\\text{DEFER}\\\text{READY}\\\text{MOUNTi}\\\text{DEFERi}\\\text{READYi}\end{matrix}\right\}\right]\left[,\left\{\begin{matrix}\underline{\text{INPUT}}\\\text{OUTPUT}\\\text{INOUT}\\\text{CHECKPOINT}\\\text{or}\\\text{CKPT}\end{matrix}\right\}\right]\left[,\left\{\begin{matrix}\text{BLOCK}\\\text{BLK}\end{matrix}\right\}=\text{XXXX}\right]$	
$ETC		[, ACT=XX] $\left[,\left\{\begin{matrix}\underline{\text{ONEREEL}}\\\text{MULTIREEL}\\\text{or}\\\text{REELS}\end{matrix}\right\}\right]\left[,\left\{\begin{matrix}\text{NOSEARCH}\\\text{SEARCH}\end{matrix}\right\}\right]$	

CARD FORMAT

1	8	16	REFERENCE PAGE
$ETC		$\left[,\left\{\begin{matrix}\underline{\text{HIGH}}\\\text{LOW}\\\text{200}\\\text{556}\\\text{800}\end{matrix}\right\}\right]$ $\left[,\left\{\begin{matrix}\underline{\text{BCD}}\\\text{BIN}\\\text{MXBCD}\\\text{MXBIN}\end{matrix}\right\}\right]$ $\left[,\left\{\begin{matrix}\underline{\text{SLABEL}}\\\text{HILABEL}\\\text{LOLABEL}\\\text{FLABEL}\end{matrix}\right\}\right]$	
$ETC		$\left[,\left\{\begin{matrix}\underline{\text{NOSEQ}}\\\text{SEQ}\\\text{or}\\\text{SEQUENCE}\end{matrix}\right\}\right]$ $\left[,\left\{\begin{matrix}\underline{\text{NOCKSUM}}\\\text{CKSUM}\end{matrix}\right\}\right]$ $\left[,\left\{\begin{matrix}\underline{\text{NOCKPTS}}\\\text{CKPTS}\end{matrix}\right\}\right]$	
$ETC		[, AFTERLABEL] $\left[,\left\{\begin{matrix}\underline{\text{SCRATCH}}\\\text{PRINT}\\\text{PUNCH}\\\text{HOLD}\end{matrix}\right\}\right]$ $\left[,\left\{\begin{matrix}\text{CYLINDER}\\\text{CYL}\end{matrix}\right\}=\text{XXX}\right]$	
$ETC		$\left[,\left\{\begin{matrix}\text{CYLCOUNT}\\\text{CYLCT}\end{matrix}\right\}=\text{XXX}\right]$ [, WRITECK] $\left[,\left\{\begin{matrix}\text{HRFP}\\\text{HRNFP}\\\text{HNRFP}\\\underline{\text{HNRNFP}}\end{matrix}\right\}\right]$	
$FTEND			11
$GROUP		[OPNCT=XX] [, BUFCT=XXX] , 'filename $_1$', . . .	24
$IBCBC	deckname	$\left[\left\{\begin{matrix}\underline{\text{NOLIST}}\\\text{LIST}\\\text{FULIST}\end{matrix}\right\}\right]$ $\left[,\left\{\begin{matrix}\underline{\text{NOREF}}\\\text{REF}\end{matrix}\right\}\right]$ $\left[,\left\{\begin{matrix}\underline{\text{DECK}}\\\text{NODECK}\end{matrix}\right\}\right]$ $\left[\left\{\begin{matrix}\underline{\text{M90}}\\\text{M94}\\\text{M94/2}\end{matrix}\right\}\right]$ $\left[,\left\{\begin{matrix}\underline{\text{XR3}}\\\text{XRn}\end{matrix}\right\}\right]$ $\left[,\left\{\begin{matrix}\underline{\text{INLINE}}\\\text{TIGHT}\end{matrix}\right\}\right]$ $\left[,\left\{\begin{matrix}\underline{\text{IOEND}}\\\text{READON}\end{matrix}\right\}\right]$ $\left[,\left\{\begin{matrix}\underline{\text{COMSEQ}}\\\text{BINSEQ}\end{matrix}\right\}\right]$	15

CARD FORMAT 1	8	16	REFERENCE PAGE
$IBFTC	deckname	$\left[\begin{Bmatrix}\underline{\text{NOLIST}}\\\text{LIST}\\\text{FULIST}\end{Bmatrix}\right] \left[,\begin{Bmatrix}\underline{\text{NOREF}}\\\text{REF}\end{Bmatrix}\right] \left[,\begin{Bmatrix}\underline{\text{DECK}}\\\text{NODECK}\end{Bmatrix}\right] \left[,\begin{Bmatrix}\underline{\text{M90}}\\\text{M94}\\\text{M94/2}\end{Bmatrix}\right] \left[,\begin{Bmatrix}\underline{\text{XR3}}\\\text{XRn}\end{Bmatrix}\right]$	13
$IBJOB		$\left[\begin{Bmatrix}\text{GO}\\\text{NOGO}\end{Bmatrix}\right] \left[\begin{Bmatrix}\text{NOLOGIC}\\\text{LOGIC}\\\text{DLOGIC}\end{Bmatrix}\right] \left[,\begin{Bmatrix}\text{NOMAP}\\\text{MAP}\end{Bmatrix}\right] \left[\left[,\begin{Bmatrix}\text{NOFILES}\\\text{FILES}\end{Bmatrix}\right]\right.$ $\left[,\begin{Bmatrix}\text{SOURCE}\\\text{NOSOURCE}\end{Bmatrix}\right] \left[\begin{Bmatrix}\text{MINIMUM}\\\text{BASIC}\\\text{LABELS}\\\text{IOEX}\\\text{FIOCS}\end{Bmatrix}\right] \left[,\begin{Bmatrix}\text{FLOW}\\\text{NOFLOW}\end{Bmatrix}\right]$	7
$IBLDR	deckname	$\left[,\begin{Bmatrix}\text{NOLIBE}\\\text{LIBE}\end{Bmatrix}\right] \left[,\begin{Bmatrix}\text{TEXT}\\\text{NOTEXT}\end{Bmatrix}\right]$	20
$IBMAP	deckname	$[\text{count}] \left[,\begin{Bmatrix}\underline{\text{LIST}}\\\text{NOLIST}\end{Bmatrix}\right] \left[,\begin{Bmatrix}\underline{\text{REF}}\\\text{NOREF}\end{Bmatrix}\right] \left[,\begin{Bmatrix}\underline{\text{DECK}}\\\text{NODECK}\end{Bmatrix}\right]$ $\left[,\begin{Bmatrix}\underline{\text{M90}}\\\text{M94}\\\text{M94/2}\end{Bmatrix}\right] \left[,\begin{Bmatrix}\underline{\text{RELMOD}}\\\text{SYMOD}\\\text{ABSMOD}\end{Bmatrix}\right] \left[,\begin{Bmatrix}\text{NO(\ \ )}\\\text{(\ \ )OK}\end{Bmatrix}\right] \left[,\begin{Bmatrix}\underline{\text{NOMFTC}}\\\text{MFTC}\end{Bmatrix}\right]$	16
$IBSYS			9
$ID			9
$IEDIT		$\left[\begin{Bmatrix}\underline{\text{SYSIN1}}\\\text{SYSxxx}\end{Bmatrix}\right] \left[,\begin{Bmatrix}\underline{\text{NOSRCH}}\\\text{SRCHn}\\\text{SCHFn}\end{Bmatrix}\right] \left[,\begin{Bmatrix}\underline{\text{NOALTER}}\\\text{ALTER}\end{Bmatrix}\right]$	11

CARD FORMAT 1	8	16	REFERENCE PAGE
$INCLUDE		$\left\{\begin{matrix}\text{exname}\\\text{deckname}\end{matrix}\right\},\ldots$	30
$INSERT		[srname] [, ORG=nnnnn]	93
$JOB			7
$LABEL		'file name' $\left[,\left\{\begin{matrix}\text{serial}\\\text{HA}\end{matrix}\right\}\right]$ [, reel] $\left[,\left\{\begin{matrix}\text{date}\\\text{days}\end{matrix}\right\}\right]$ [, name]	23
$NAME		$\left[\left\{\begin{matrix}\text{deckname (exname)=exname}\\\text{exname=exname}\\\text{`deckname (filename)'=`filename'}\\\text{`filename'=`filename'}\end{matrix}\right\}\right]$	25
$OEDIT		$\left[\left\{\begin{matrix}\underline{\text{SYSOU1}}\\\text{SYSxxx}\end{matrix}\right\}\right]$ $\left[,\left\{\begin{matrix}\underline{\text{NOPREST}}\\\text{PREST}\end{matrix}\right\}\right]$ $\left[,\left\{\begin{matrix}\underline{\text{NOCPR}}\\\text{CPREST}\end{matrix}\right\}\right]$	11
$OMIT		$\left\{\begin{matrix}\text{exname}\\\text{deckname (exname)}\end{matrix}\right\},\ldots$	24
$ORIGIN		$\begin{matrix}\text{logical}\\\text{origin}\end{matrix}\left[,\begin{matrix}\text{absolute}\\\text{origin}\end{matrix}\right]$ $\left[,\left\{\begin{matrix}\underline{\text{SYSUT2}}\\\text{SYSxxx}\end{matrix}\right\}\right]$ $\left[,\left\{\begin{matrix}\underline{\text{NOREW}}\\\text{REW}\end{matrix}\right\}\right]$	30

CARD FORMAT

1	8	16	REFERENCE PAGE
$PATCH	cxxxxx	instr. 1, instr. 2, . . .	45
$PAUSE			9
$POOL		$\left[\begin{Bmatrix}\text{BLOCK}\\\text{BLK}\end{Bmatrix}=\text{XXXX}\right]$ [, BUFCT=XXX] , 'filename' , . . .	24
$REPLACE		srname[, ORG=nnnnn]	93
$SIZE		//=n	25
$STOP			9
$USE		deckname (exname) , . . .	24

Appendix B: Control Card Check List

	SOURCE LANGUAGE PROGRAMS			RELOCATABLE BINARY PROGRAMS	
	COBOL	FORTRAN	MAP		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
$IBFTC		x			Precedes each FORTRAN deck
$FTEND		o			Required after each FORTRAN deck to be compressed into Prest form
$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 or 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 section to be included in a link
*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 Processor Output

Specifications on the $IBJOB card. Component control cards and the $OEDIT card cause punched output and listing output.

Punched Output

The following punched output can be produced:

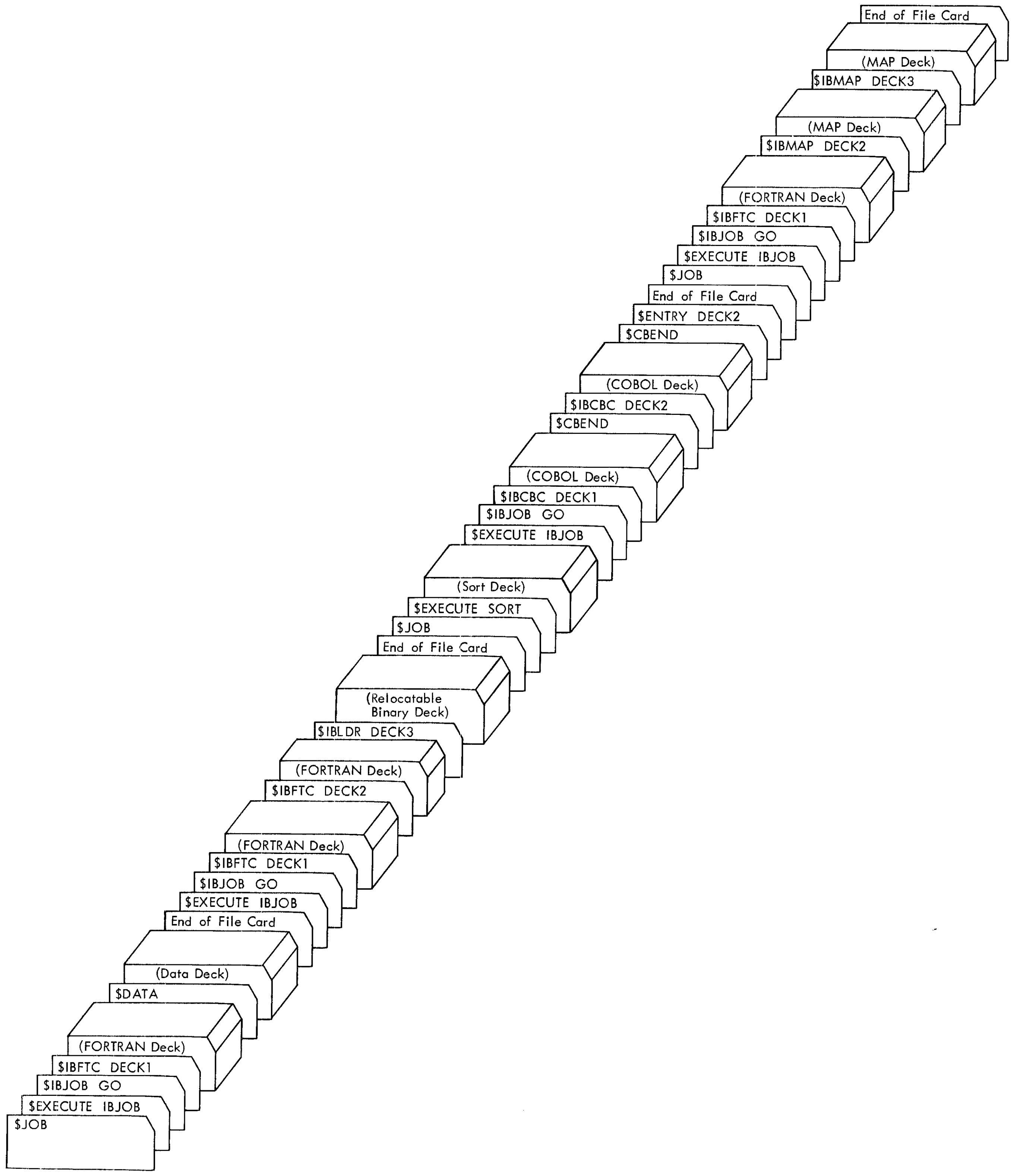
1. Binary object program deck, unless NODECK is specified.
2. Compiler Prest deck, if CPREST is specified.
3. Symbolic Prest deck, if PREST is specified.

Listing Output

The following listing output can be produced:

1. Source program listing.
2. Diagnostic and on-line messages.
3. Assembly listing, if LIST or FULIST is specified.
4. Core storage map, if MAP is specified.
5. Cross-reference table of object program symbols, if REF is specified.
6. Cross-reference table of program sections, if LOGIC or DLOGIC is specified.
7. Listing of input/output unit assignments and mounting instructions, if FILES is specified.

Appendix D: Sample Control Card Deck



Index

C28-6275-2

Printed in U.S.A. C28-6275-2

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601

Reader's Comments

IBM 7090/7094 IBSYS OPERATING SYSTEM: IBJOB PROCESSOR
Form C28-6275-2

From

Name ____________________________________

Address ____________________________________

Your comments regarding the completeness, clarity, and accuracy of this publication will help us improve future editions. Please check the appropriate items below, add your comments, and mail.

	YES	NO
Does this publication meet the needs of you and your staff?	____	____
Is this publication clearly written?	____	____
Is the material properly arranged?	____	____

If the answer to any of these questions is "NO, " be sure to elaborate.

How can we improve this publication? Please answer below.

☐ Suggested Addition (Page , Timing Chart, Drawing, Procedure, etc.)

☐ Suggested Deletion (Page)

☐ Error (Page)

COMMENTS:

No Postage Necessary if Mailed in U.S.A.

STAPLE STAPLE

FOLD FOLD

FIRST CLASS
PERMIT NO. 81

POUGHKEEPSIE, N.Y.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

IBM CORPORATION
P.O. BOX 390
POUGHKEEPSIE, N.Y.

ATTN: PROGRAMMING SYSTEMS PUBLICATIONS
DEPARTMENT D91

CUT ALONG LINE

FOLD FOLD

2/64:10M-EP-108

STAPLE STAPLE