File No. 7090-24
Form C28-6391-0

IBM Systems Reference Library

IBM 7090/7094 IBSYS Operating System

Version 13

COBOL Language

The COBOL language (Common Business Oriented Language) is a programming language closely related to English. This publication describes the COBOL language used with the 7090/7094 Data Processing System. Programs written in the COBOL language are translated by the 7090/7094 COBOL Compiler (7090-CB-806), which is a component of the 7090/7094 IBJOB Processor, a subset of the IBSYS Operating System (Version 13).

Part I of this publication describes the 7090/7094 COBOL language so that programmers with no prior knowledge of COBOL can plan and write programs in the language. Part II of this publication contains examples, programming pointers, control card information, and error messages.

Preface

This publication describes the 7090/7094 COBOL language for both experienced and inexperienced COBOL programmers. Part I of the publication explains the rules of the 7090/7094 COBOL language. The four divisions of a COBOL program are discussed in the order in which they appear in a program: IDENTIFICATION DIVISION, ENVIRONMENT DIVISION, DATA DIVISION, and PROCEDURE DIVISION. Part II contains supplemental information on the use of the language. It includes a description of control cards needed to run a program, explanation of error messages produced by the compiler, examples of some of the language statements, and a list of programming pointers. Reference material that is frequently used has been put in the appendixes. There is also a glossary of terms used in the publication.

The COBOL Compiler is a component of the IBJOB Processor. The publication *IBM 7090/7094 IBSYS Operating System, IBJOB Processor*, Form C28-6389, contains a description of the IBJOB Processor and includes machine requirements.

The following publications contain useful supplemental material.

Common Business Oriented Language (COBOL) General Information, Form F28-8053

IBM 7090/7094 IBSYS Operating System, Input/Output Control System, Form C28-6345

IBM 7090/7094 IBSYS Operating System, IBJOB Processor Debugging Package, Form C28-6393

This publication, Form C28-6391-0, updates the material in Forms J28-6260-1 and J28-6260-2 and their associated technical newsletters Forms N28-0040, N28-0052, N28-0070, N28-0092, N28-0092-1, and N28-0103 for users of Version 13 of the IBSYS Operating System.

A new section including control card information and programming examples has been added. Most of the text has been revised. New technical material, indicated by a vertical line in the margin to the left of the change, updates the paragraph on I-O-CONTROL under "Environment Division" and the description of the DISPLAY and ENTER verbs under "Procedure Division."

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, 1271 Avenue of the Americas, New York 20, N. Y. 10020

Acknowledgement

The material in this bulletin has been prepared by altering the official COBOL-61 manual of the Department of Defense. Many of the elective features and paragraphs of general commentary have been eliminated; and descriptions of certain adaptations of COBOL to the IBM 7090 and 7094 computers have been added. In accordance with the requirements of that manual, the following extract is presented for the information and guidance of the user.

"This publication is based on the COBOL System developed in 1959 by a committee composed of government users and computer manufacturers. The organization participating in the original development were:

Air Materiel Command, United States Air Force
Bureau of Standards, United States Department of Commerce
Burroughs Corporation
David Taylor Model Basin, Bureau of Ships, United States Navy
Electronic Data Processing Division, Minneapolis-Honeywell Regulator Company
International Business Machines Corporation
Radio Corporation of America
Sylvania Electric Products, Inc.
UNIVAC Division of Sperry Rand Corporation.

"In addition to the organizations listed above, the following other organizations participated in the work of the Maintenance Group:

Allstate Insurance Company
The Bendix Corporation, Computer Division
Control Data Corporation
E. I. du Pont de Nemours and Company
General Electric Company
General Motors Corporation
Lockheed Aircraft Corporation
The National Cash Register Company
Philco Corporation
Standard Oil Company (New Jersey)
United States Steel Corporation

"This COBOL-61 manual is the result of contributions made by all of the above-mentioned organizations. No warranty, expressed or implied, is made by any contributor or by the committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection therewith.

"It is reasonable to expect that many improvements and additions will be made to COBOL. Every effort will be made to insure that improvements and corrections will be made in an orderly fashion, with due recognition of existing users' investments in the programming. However, this protection can be positively assured only by individual implementors.

"Procedures have been established for the maintenance of COBOL. Inquiries concerning the procedure and methods for proposing changes should be directed to the Executive Committee of the Conference on Data Systems Languages.

"The authors and copyright holders of the copyrighted material used herein: FLOW-MATIC* (Trademark of Sperry Rand Corporation) *Programming for the UNIVAC* I and II, Data Automation Systems*© 1958, 1959, Sperry Rand Corporation; *IBM Commercial Translator*, Form No. F28-8013, copyrighted 1959 by IBM; FACT DSI 27 A5260-2760, copyrighted 1960 by Minneapolis-Honeywell, have specifically authorized the use of this material, in whole or in part, in the COBOL-61 specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications.

"Any organization interested in reproducing the COBOL report and initial specifications, in whole or in part, using ideas taken from this report or utilizing this report as the basis for an instruction manual or any other purpose is free to do so. However, all such organizations are requested to reproduce this section as part of the introduction to the document. Those using a short passage, as in a book review, are requested to mention "COBOL" in acknowledgment of the source, but need not quote the entire section."

*Trademark of Sperry Rand Corporation

Contents

List of Illustrations

PART I

Introduction

A COBOL programming system consists of the COBOL language (or a subset) and a compiler. The COBOL language was designed for use with computers that differ in size and logical structure. A COBOL compiler is a program that adapts the language to the computer on which the object program is to be run. The 7090/7094 COBOL compiler translates a 7090/7094 COBOL language source program into a mnemonic programming language that is closer to machine language than COBOL. This language is, in turn, translated into the machine language that can be executed on the 7090/7094 computers. These operations are directed by control cards (explained in Part II of this publication).

Many detailed operations, which are difficult or lengthy to program in machine language or a language that is close to machine language, can be generated by several instructions written in the COBOL language. For instance, if externally stored information is to be used in a program, it is necessary to determine when to bring it into the computer, where to put it, and what to do in the event of an error or malfunction in transmission. One procedure statement takes care of all of this in a COBOL program.

The COBOL language is very much like English. The restrictions imposed on the language are necessary to ensure efficient compilation and assembly. Since the language is so much like English, a COBOL program is self-documenting; the logic of the program can be understood from the program and additional documentation is unnecessary.

Divisions of a COBOL Program

There are four divisions of a COBOL program: the IDENTIFICATION DIVISION, the ENVIRONMENT DIVISION, the DATA DIVISION, and the PROCEDURE DIVISION. The statements to solve the problem are given in the PROCEDURE DIVISION. The other divisions provide information so that the statements in the PROCEDURE DIVISION can be executed.

The IDENTIFICATION DIVISION identifies the program with information such as the name of the program and the programmer, the date, and general remarks.

The ENVIRONMENT DIVISION relates the physical machine configuration to the program. Information that is peculiar to a given computer or that is associated with the physical machine configuration is given in this division. One of the major functions of the ENVIRONMENT DIVISION is to specify the input-output units (magnetic tape units, card reader, card punch, printer, etc.) to be used by the program. Therefore, if a program written for a computer with one configuration is to be run on a computer with a different configuration, this is the division that is changed.

The DATA DIVISION contains a description of data used in the program. The familiar concept of files and records is retained in the COBOL data organization. An organization of data related to a particular subject is called a logical record. A file is a group of logical records stored outside the computer. For example, an employee file can be composed of a payroll record and a personnel record for each employee in the company.

A description of the physical characteristics of each file used in the program must be given in this division. Some of the information that can be provided is recording mode, density, and block size.

Each logical record in the file must also be described in the DATA DIVISION. The programmer describes the type of logical record rather than each individual record. For example, for the employee file, only the two types of logical records, payroll records and personnel records, are described. The entries form a dummy description of an area of storage that is filled with the values in the record when it is moved to the area.

In addition to the descriptions of files and records, the DATA DIVISION must contain a detailed description of information used in the program that is not organized into files, for example, tables and constants.

The PROCEDURE DIVISION contains instructions or statements for solving the problem. Two types of statements can be used in this division: imperative statements to transmit data, perform arithmetic operations, direct the compiler, etc.; and conditional statements to test the sign of a number, the on or off status of an entry key, etc.

Notation Used in this Publication

Throughout this publication, basic formats for entries are given as they should appear on a COBOL program sheet. The rules for using the program sheet are given in "Appendix A." The following notation is used in the formats:

1. All upper case words that are underlined are required.

2. All upper case words that are not underlined are used for readability only and may be omitted.

3. All lower case words represent information that must be supplied by the programmer. The nature of the information required is indicated in each case.

4. When material is enclosed in braces { }, one, and only one, of the enclosed items is required; the others are to be omitted. The choice is made by the programmer.

5. Material enclosed in brackets [ ] represents an option and may be included or omitted.

6. Sometimes part of a format may be repeated. This is indicated by three dots (an ellipsis) following the part to be repeated. The dots apply to the last complete element preceding them. For example, if a group of items is enclosed in brackets and three dots precede the right bracket, everything in the brackets must be repeated.

7. Punctuation, where shown, is obligatory.

8. When items are written in a series, they may be separated by a space, by a comma followed by a space, by the word AND, or by a comma followed by the word AND.

9. In some cases where many choices are available, the formats are separated into numbered options.

The IDENTIFICATION DIVISION is used to identify or label the program and to provide comments about it. The information becomes a part of the listing, but has no effect on the program. The format for the IDENTIFICATION DIVISION is:

```
{IDENTIFICATION DIVISION.}
{ID DIVISION.            }
    PROGRAM-ID. program-name.
    [AUTHOR. author-name.]
    [INSTALLATION. any sentence or group of sentences.]
    [DATE-WRITTEN. any sentence or group of sentences.]
    [DATE-COMPILED. any sentence or group of sentences.]
    [SECURITY. any sentence or group of sentences.]
    [REMARKS. any sentence or group of sentences.]
```

The PROGRAM-ID appears at the beginning of the output listing. *Program-name* is the identifying name of the program. It may be composed of from 1 to 30 characters chosen from the letters A through Z, the numbers 0 through 9, and the hyphen. It may not contain embedded blanks and may neither begin nor end with a hyphen.

Figure 1 shows sample entries in the IDENTIFICATION DIVISION.

IBM

COBOL PROGRAM SHEET

Form No. X28-1464
Printed in U.S.A.

PAGE 010 | PROGRAM TEST PROGRAM | SYSTEM IBM-7094 | SHEET 1 OF 8
PROGRAMMER A.F.A. | DATE 5/28 | IDENT. TEST 01

SERIAL	CONT	A / B
010		IDENTIFICATION DIVISION.
020		PROGRAM-ID. TEST-PROGRAM.
030		AUTHOR. ANDREW FULLER ASTON.
040		INSTALLATION. WASHINGTON CORPORATION.
050		DATE-WRITTEN. MAY 28, 1965.
060		REMARKS. THIS IS TEST 1 OF 6 TESTS.

Figure 1. Sample Entries in the IDENTIFICATION DIVISION

One of the major advantages of the COBOL language is that a program written for one computer can be run on a different computer with only a few changes in the program. Information about the physical characteristics of the computer is given in the ENVIRONMENT DIVISION, so that a change in computers entails major changes in the ENVIRONMENT DIVISION only.

The ENVIRONMENT DIVISION is composed of two sections: the CONFIGURATION SECTION and the INPUT-OUTPUT SECTION. Entries in these sections are optional. There are three paragraphs in the CONFIGURATION SECTION. The SOURCE-COMPUTER paragraph identifies the computer on which the program is to be compiled. The OBJECT-COMPUTER paragraph identifies the computer on which the compiled program is to be run. The SPECIAL-NAMES paragraph assigns names to the entry keys on the computer so they can be referred to in the program.

Configuration Section

The CONFIGURATION SECTION specifies the computer on which the program is to be compiled and the computer on which the program is to be run. Names may also be assigned to the entry keys on the computer in this section.

There are two paragraphs in the INPUT-OUTPUT SECTION. The FILE-CONTROL paragraph assigns files to input-output devices, and the I-O-CONTROL paragraph provides for taking checkpoints for rerun purposes and causes block sequence numbers or checksums to be written or checked.

The following list shows the order of the entries in the ENVIRONMENT DIVISION. Entries may be omitted if they are not needed in the program.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.
OBJECT-COMPUTER.
SPECIAL-NAMES.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
I-O-CONTROL.

Figure 2 shows sample coding in the ENVIRONMENT DIVISION. Detailed explanations are given under the names of the entries.

SOURCE-COMPUTER Paragraph

This entry identifies the computer on which the program is to be compiled. The format is:

```
[ SOURCE-COMPUTER. {IBM-7090.} ]
                   {IBM-7094.}
```

IBM

COBOL PROGRAM SHEET

Form No. X28-1464 Printed in U.S.A.

PAGE 3	PROGRAM TEST PROGRAM		SYSTEM IBM-7094	SHEET 2	OF 8
020	PROGRAMMER A.F.A.		DATE 5/28	IDENT.	TEST 01

SERIAL	CONT.	A B	
010		ENVIRONMENT DIVISION.	
020		CONFIGURATION SECTION.	
030		SOURCE-COMPUTER. IBM-7094.	
040		OBJECT-COMPUTER. IBM-7094.	
050		SPECIAL-NAMES. KEY 3 IS INTERRUPT ON STATUS IS MESSAGE1	
060		OFF STATUS IS MESSAG2, KEY 22 ON STATUS IS CONTINUATION.	
070		INPUT-OUTPUT SECTION.	
080		FILE-CONTROL. SELECT EMPLOYEE-FILE ASSIGN TO TAPE-UNITS FOR	
090		MULTIPLE REEL SELECT MASTER-INPUT ASSIGN TO TAPE-UNIT	
100		SELECT OUTPUT-FILE RENAMING EMPLOYEE-FILE ASSIGN TO	
110		TAPE-UNIT.	
120		I-O-CONTROL. APPLY CHECKSUM ON EMPLOYEE-FILE MASTER-INPUT	
130		RERUN ON CHECKPOINT-UNIT EVERY BEGINNING OF REEL OF	
140		ALL FILES.	

Figure 2. Sample Coding in the ENVIRONMENT DIVISION

OBJECT-COMPUTER Paragraph

This entry identifies the computer on which the compiled program is to be executed. The format is:

```
[ OBJECT-COMPUTER.  { IBM-7090. }  ]
                    { IBM-7094. }
```

SPECIAL-NAMES Paragraph

This entry is used to assign names to the ON and/or OFF status of the entry keys on the computer. The switch status test in the PROCEDURE DIVISION can be used to determine whether an entry key is ON or OFF during the program. The machine operator sets the entry keys when the program is to be run; by definition, an entry key is ON if it is down and OFF if it is up. An installation may have restrictions as to which, if any, of the entry keys may be used by the program. The format of the SPECIAL-NAMES paragraph is:

```
[ SPECIAL-NAMES.  KEY { S  } [IS mnemonic-name]
                      { 1  }
                      {  .  }
                      {  .  }
                      { 35 }

        { ON STATUS IS switch-status-name-1  }
        { OFF STATUS IS switch-status-name-2 }

      [ { ON STATUS IS switch-status-name-3  } ]
        { OFF STATUS IS switch-status-name-4 }

[ KEY { S  }  ...  ]  .  ]
      { 1  }       
      {  .  }      
      {  .  }      
      { 35 }
```

The 36 entry keys that may be tested are numbered S, 1 through 35. A key may be designated only once in the paragraph but the ON and OFF positions can each be assigned two switch-status-names. The order of the ON and OFF clauses is not significant.

The optional clause, IS *mnemonic-name*, is used for readability only. This name cannot be referred to in the PROCEDURE DIVISION.

Input-Output Section

The INPUT-OUTPUT SECTION specifies the peripheral devices that are to be used in the program to provide data to the computer and to receive processed data from the computer. This section is also used to specify the places in the program where checkpoints are to be taken if the program is to be rerun and to request the formation and checking of block sequence numbers and checksums.

FILE-CONTROL Paragraph

Information that is used or developed by the program may be stored externally. File description entries in the DATA DIVISION name the files into which the information is arranged and specify their physical characteristics. The FILE-CONTROL paragraph assigns the files (by the name given in the file description entries) to input-output devices.

There are three forms of the FILE-CONTROL paragraph. Option 1 is the general form. Options 2 and 3 assign files to specific input-output devices.

The three formats of the FILE-CONTROL paragraph are:

```
Option 1.
[ FILE-CONTROL.  SELECT file-name-1
                        [RENAMING file-name-2]
                 { 1 TAPE-UNIT                          }
                 { 1 TAPE-UNIT FOR MULTIPLE REEL        }
      ASSIGN TO  { 2 TAPE-UNITS FOR MULTIPLE REEL       }
                 { CARD-READER                          }
                 { PRINTER                              }
                 { CARD-PUNCH                           }
  [SELECT...] . ]
```

```
Option 2.
[ FILE-CONTROL.  SELECT file-name-1
                        [RENAMING file-name-2]
                 { symbolic-tape-unit-name
                                [FOR MULTIPLE REEL]     }
                 { symbolic-tape-unit-name-1
                                symbolic-tape-unit-name-2
                        FOR MULTIPLE REEL               }
      ASSIGN TO  { symbolic-card-unit-name              }
                 { system-unit-name
                                [FOR MULTIPLE REEL]     }
                 { system-unit-name-1  system-unit-name-2
                        FOR MULTIPLE REEL               }
                 { NONE                                 }
  [SELECT...] . ]
```

```
Option 3.
[ FILE-CONTROL.  SELECT file-name-1
                        [RENAMING file-name-2]
                 { hypertape-unit-name
                                [FOR MULTIPLE REEL]     }
      ASSIGN TO  { hypertape-unit-name-1
      HYPERTAPE                 hypertape-unit-name-2
                        FOR MULTIPLE REEL               }
  [SELECT...] . ]
```

Each file must be assigned to an input-output device with a SELECT entry in the FILE-CONTROL paragraph. The word SELECT identifies the beginning of information for each file. If a file is used for both input and output, there must be two SELECT entries for it.

File-name-1 is the name of the file being assigned to a peripheral device. If the RENAMING clause is not used, this is the name given to the file in the file description entry in the DATA DIVISION. If the RENAMING clause is used, *file-name-1* is the name of a file that has the same description as *file-name-2*. There is no separate file description entry for *file-name-1;* however, *file-name-2* must have a separate SELECT entry. Two files that share a file description entry share a storage area; therefore, they must not be in OPEN status at the same time. That is, if, in the PROCEDURE DIVISION, an OPEN statement is executed for one of the files, a CLOSE statement must be executed for that file before an OPEN statement is executed for the other file.

File-name-1 and *file-name-2* are formed according to the rules for names; that is, they must consist of from 1 to 30 characters chosen from the letters A through Z, the numbers 0 through 9, and the hyphen. At least one of the characters must be alphabetic. The names must not contain embedded blanks and must neither begin nor end with a hyphen.

Only the ASSIGN clause differs in the three options. MULTIPLE REEL must be specified in the ASSIGN clause in all three options if the file is on two or more reels of magnetic tape. A file may be assigned to one or two, but not more than two, tape units. If a file is assigned to two tape units, the units are used alternately for successive reels.

When option 1 is used, the programmer has no control over which physical unit is to be assigned; for example, if TAPE-UNIT is specified, any one of the tape units on any one of the channels may be assigned by the operating system.

Option 2 is used to assign files to specific magnetic tape units and unit record equipment. If ASSIGN TO NONE is specified in option 2, the file is assumed to be unavailable during the running of the program. Option 3 is used to assign files to specific Hypertape units.

Files may be assigned to specific magnetic tape units, unit record equipment, or Hypertape units by specifying the device to be used after the words ASSIGN TO or ASSIGN TO HYPERTAPE. The device may be specified as a symbolic unit, an intersystem unit, or a system unit.

Symbolic Units: Files may be assigned to symbolic units with a code that specifies the device and/or channel and/or unit. Figure 3 shows the codes that are used, the range of values of the codes, and their interpretation.

Intersystem Units: Intersystem units are used for files that provide data to related programs. The unit used for an output file in one program may be reserved for an input or an output file in a different program. Figure 4 shows the codes used for symbolic

Symbolic Tape Units

Codes	Range of Values	Interpretation
M	M = II or IV (Denotes an IBM 729 II or IV Magnetic Tape Unit.)	Any available tape unit of model number M is assigned to the file.
X	X = A, B, . . . , H (Denotes one of the real channels A, B, . . . , H.)	Any available tape unit on physical channel X is assigned to the file.
P	P = S, T, . . . , Z (Denotes a symbolic, i.e., unspecified, physical channel S, T, . . . , Z.)	All files in the program having symbolic channel P designation are assigned to the same channel, if possible.
X(k)	X = A, B, . . . , H k = 0, 1, . . . , 9 (k denotes one of the unit numbers 0, 1, . . . , 9. X denotes one of the real channels.)	The kth available tape unit on the specified channel X is assigned to the file. Note that the parentheses are required.
PM	P = S, T, . . . , Z M = II or IV (P denotes the symbolic channel, and M denotes the model of the tape unit.)	Any available tape unit of model type M on the specified symbolic channel P is assigned to the file.
P(k)M	P = S, T, . . . , Z k = 0, 1, . . . , 9 M = II or IV (P denotes the symbolic channel, k denotes the unit number, and M denotes the model.)	Any available tape unit on the symbolic channel P having model number M is assigned to the file. The (k) in this usage indicates the order of preference for the channel so that if the number of available tape units on the channel is less than the total requested for the channel, those with lower numbers are assigned to the same channel. Note that the parentheses are required.

Symbolic Unit Record Equipment

Code	Range of Values	Interpretation
RDX	X = A, B, . . . , H (Denotes the real channels.)	The card reader on channel X is assigned to the file.
PRX	X = A, B, . . . , H (Denotes the real channels.)	The printer on channel X is assigned to the file.
PUX	X = A, B, . . . , H (Denotes the real channels.)	The card punch on channel X is assigned to the file.

Symbolic Hypertape Units

Codes	Range of Values	Interpretation
XHK/I	X = A, B, . . . , H H = H K = 0, 1, . . . , 9 I = 0 or 1. (X denotes one of the real channels A, B, . . . , H. H denotes Hypertape. K denotes one of the unit numbers 0, 1, . . . , 9. I denotes interface 0 or 1.)	The Kth available Hypertape unit on channel X with interface I is assigned to the file. If the interface is to be 0, the designation /I may be omitted.

Figure 3. Codes Used to Assign Files to Symbolic Units

Code	Range of Values	Interpretation
I	I = J, K, . . . , Q (I denotes a symbolic intersystem, i.e., unspecified, physical channel J, K, . . . , Q.)	The file is assigned to intersystem channel I.
IM	I = J, K, . . . , Q M = II or IV (I denotes an intersystem channel, and M denotes the model of the magnetic tape unit.)	The file is assigned to intersystem channel I, model M. The model must not be given for an intersystem input unit.
I(k)	I = J, K, . . . , Q k = 0, 1, . . . , 9 (I denotes an intersystem channel, and k denotes the unit.)	The file is assigned to the kth available unit on intersystem channel I. Note that the parentheses are required.

NOTE: An R may be added to any of the codes to indicate to the operating system that reserve status for the unit ends after the current job is complete.

Figure 4. Codes Used to Assign Files to Symbolic Intersystem Units

intersystem units, their range of values, and their interpretation.

System Units: System names have been given to devices that have special functions within the IBSYS Operating System. For example, the system input unit has the name SYSIN1. When the system tape is distributed to the installation, SYSIN1 is the name for channel A, unit 2. Although the installation may change the machine configuration, the system input unit is still named SYSIN1. Files can be assigned to peripheral devices by system names. Figure 5 shows the system name, the physical device to which the system unit was assigned when the system tape was distributed to the installation, and the interpretation of the system name.

Altering Unit Assignments: When a COBOL program is compiled, the information about the files provided in the FILE-CONTROL paragraph in the ENVIRONMENT DIVISION and in the file description entries in the DATA DIVISION is combined to form a complete description of the files. When the program is loaded, this information can be altered by a $FILE loader control card. The $FILE control card is placed after the $IBLDR control card in the binary deck, which is output from the compilation if requested on the $IBCBC control card. The rules for altering unit assignments are:

1. Files assigned to tape units (including files designated ASSIGN TO NONE) cannot be reassigned to unit record equipment.

2. Files assigned to unit record equipment cannot be reassigned to tape units.

3. Files assigned to peripheral equipment by system names that allow either tape or unit record equipment (SYSIN1, SYSOU1, and SYSPP1) may be changed to any other tape or unit record equipment. The description of the file must, however, conform to the type of equipment being used.

System Name	Assignments in the Distributed Version Only	Interpretation
SYSIN1	A2	The file is assigned to the current system input unit. The physical unit may be determined by the machine operator and may be a card reader or a tape unit.
SYSOU1	B1	The file is assigned to the current system output unit. The physical unit may be a printer or a tape unit.
SYSPP1	B2	The file is assigned to the current system peripheral punch unit. The physical unit may be a card punch or a tape unit.
SYSUTk	k = 1 A3 = 2 B3 = 3 A4 = 4 B4 5-9 not assigned	The file is assigned to system utility tape k where k = 1 through 9.
SYSLBk	k = 1 A1 2-4 not assigned	The file is assigned to system library tape k where k = 1, 2, 3 or 4.
SYSCRD	on-line card reader	The file is assigned to the system card reader.
SYSPRT	on-line printer	The file is assigned to the system printer.
SYSPCH	on-line card punch	The file is assigned to the system card punch.
SYSCKk	not assigned	The file is assigned to the system checkpoint unit k where k = 1 or 2.

NOTE: The $FILE loader card must be used to assign files by system name to 7340 Hypertape units. When this is done, the density specification in the RECORDING MODE clause in the file description entry is ignored. If a file is assigned to two units and the first is a Hypertape unit, the second must also be a Hypertape unit.

Figure 5. System Names Used to Assign Files to System Units

Figure 6 shows the relationship between the forms of the ASSIGN TO clause in the FILE-CONTROL paragraph and the unit assignment fields on the $FILE card. The $FILE control card is described in *IBM 7090/7094 IBSYS Operating System; IBJOB Processor*, Form C28-6389.

I-O-CONTROL Paragraph

This optional paragraph has two clauses. The first clause (the RERUN clause) specifies that *checkpoints* are to be taken. A checkpoint is a reference point in the program. The contents of core storage are recorded on magnetic tape, which can be read back into core storage to restart the program from an intermediate point.

The second clause (the APPLY clause) is used to request the formation and checking of the *block sequence number* or of the *checksum* of a block of logical records on a file. The block sequence number shows the number of the block on the current reel of the file, that is, the first block on the current reel, the second block on the current reel, etc. The checksum, formed by logical addition of the data in the block, is used to check that the data has been transmitted accurately. The block sequence number and/or the checksum is

Form of ASSIGN TO Clause	Contents of Associated $FILE Card Fields		
	Primary Unit	Secondary Unit	Multireel Field
1 TAPE-UNIT	omitted	omitted	omitted
1 TAPE-UNIT MULTIPLE REEL	omitted	omitted	REELS
2 TAPE-UNITS MULTIPLE REEL	omitted	omitted	REELS
CARD-READER	CRD	omitted	omitted
CARD-PUNCH	PCH	omitted	omitted
PRINTER	PRT	omitted	omitted
symbolic-tape-unit-name	symbolic-tape-unit-name	omitted	omitted
Example: A	A	omitted	omitted
symbolic-tape-unit-name FOR MULTIPLE REEL	symbolic-tape-unit-name	omitted	REELS
Example: TIV MULTIPLE REEL	TIV	omitted	REELS
symbolic-tape-unit-name-1 symbolic-tape-unit-name-2 FOR MULTIPLE REEL	symbolic-tape-unit-name-1	symbolic-tape-unit-name-2	REELS
Example: C(3), C(4) MULTIPLE REEL	C(3)	C(4)	REELS
symbolic-card-unit-name	symbolic-card-unit-name	omitted	omitted
Example: RDA	RDA	omitted	omitted
system-unit-name	abbreviated system-unit-name	omitted or may be filled in automatically	
Example: SYSIN1*	IN1	IN2	REELS
system-unit-name FOR MULTIPLE REEL	abbreviated system-unit-name	omitted or may be filled in automatically	REELS
Example: SYSUT1 MULTIPLE REEL	UT1	omitted	REELS
system-unit-name-1 system-unit-name-2	abbreviated system-unit-name-1	abbreviated system-unit-name-2	REELS
Example: SYSUT2, SYSUT3 FOR MULTIPLE REEL	UT2	UT3	REELS
NONE	NONE	omitted	omitted

*In the example, this particular file automatically takes on the MULTIREEL characteristic of the system unit. This is also true for SYSOU1 and SYSPP1 which are system MULTIREEL files.

Figure 6. File Assignment Table — $FILE Control Card

written in a block sequence word, which is appended by the compiler to each block of logical records.

The format of the I-O-CONTROL paragraph is:

```
I-O-CONTROL.
{RERUN [ON CHECKPOINT-UNIT]
                              EVERY BEGINNING
     OF REEL OF {ALL FILES                  }
                {INPUT FILES                }
                {OUTPUT FILES               }
                {file-name-1 [file-name-2...]}
 APPLY {SEQUENCE-CHECK} ON
       {CHECK-SUM     }
                {ALL FILES                  }
                {INPUT FILES                }
                {OUTPUT FILES               }
                {file-name-1 [file-name-2...]}
[{RERUN...}].
 {APPLY...}
```

The RERUN clause, which requests the taking of checkpoints, may be given for all input and/or output files or for only those specified by *file-name-1, file-name-2,* etc. If ON CHECKPOINT-UNIT is specified, checkpoints are written on the unit provided as the standard checkpoint unit each time a real switch occurs. If ON CHECKPOINT-UNIT is not specified, checkpoints are written, each time a reel switch occurs, following the header label on the new reel of tape for labeled output files. If the file is not a labeled output file, ON CHECKPOINT-UNIT must be specified.

The APPLY clause is used to request the checking of block sequence numbers (SEQUENCE-CHECK) or the checksums (CHECK-SUM) for input files and the writing of the block sequence numbers or checksums for output files. The formation or checking of block sequence numbers or checksums may be requested for all input and/or output files or for only those specified by *file-name-1, file-name-2,* etc. The following rules apply to the use of the APPLY clause:

1. The clause may be used only for files recorded in the binary mode.

2. If CHECK-SUM is specified, SEQUENCE-CHECK must also be specified.

3. If an input file contains block sequence words, they must be accounted for in determining the size of the blocks in the file. When the APPLY clause is used, the compiler automatically adds one word to the block size. If the APPLY clause is not used and there are block sequence words on the file, the user must add one word to the block size specified in the BLOCK CONTAINS clause of the file description entry.

If the input-output system detects an error in the checksum or block sequence number, a message indicating the type of error is written, a core-storage dump is taken, and execution of the program is terminated. It is possible to avoid taking the dump and terminating execution by specifying alternate procedures in the declaratives section of the PROCEDURE DIVISION. An example is given in Part II of this publication.

The I-O-CONTROL paragraph is optional, but it must follow the FILE-CONTROL paragraph when it is used.

Organization of Data

A *file* is the largest body of related information in a COBOL program. A file may consist of any number of *logical records* composed of *group items* and *elementary items.*

An elementary item is a piece of data that is never further divided. For example, INTEREST-RATE can be the name of an elementary item if it contains no subdivisions. Several elementary items that are logically connected can be subdivisions of a group item. For example, DATE can be the name of a group item containing elementary items YEAR, MONTH, and DAY. The term *data-item* as used in this publication refers to both group items and elementary items.

Elementary items and group items can be organized to form logical records. For example, a payroll record can contain data-items such as the name of an employee, his rate of pay, and the number of his dependents. The series of records, one for each employee, form a file. All of the records have the same format, that is, contain the same data-items, but the items have different values for each employee. One file might contain a series of payroll records and a series of personnel records, or there might be two separate files with only one type of record on each file.

Structure of the Data Division

The DATA DIVISION contains descriptions of data used in a program. It is divided into three sections: the FILE SECTION, the WORKING-STORAGE SECTION, and the CONSTANT SECTION.

All data that is stored externally in files, for example, on magnetic tape, is described in the FILE SECTION. Information that is developed and stored in the computer is described in the WORKING-STORAGE SECTION and CONSTANT SECTION.

There are two types of entries in the DATA DIVISION: *file description entries* and *data-item description entries.* File description entries appear only in the FILE SECTION. They describe the physical characteristics of the files. Data-item description entries are used in all three sections. They describe data as it appears in core storage.

Since data-item descriptions appear in all three sections, the general information about them is given before the specific rules pertaining to their use in each section. The file description entry explanation and restrictions for data-item description entries are given under the name of the section to which they apply.

The following list shows the required order of entries in the DATA DIVISION. If a section is omitted, its name need not appear in the source program.

```
DATA DIVISION.
FILE SECTION.
file description entry.
data-item description entries.
file description entry.
data-item description entries.
      .
      .
WORKING-STORAGE SECTION.
independent working-storage data-item description entries.
organized working-storage data-item description entries.
CONSTANT SECTION.
independent constant data-item description entries.
organized constant data-item description entries.
```

Data-Item Description Entry

Data-item descriptions are used in all three sections of the DATA DIVISION to describe data as it appears in core storage. In the FILE SECTION they are used to describe logical records which are stored externally in files.

Data-item descriptions of internally stored data are in the WORKING-STORAGE SECTION and the CONSTANT SECTION. For example, if a record is to be moved to a work area for processing, the work area can be described in the WORKING-STORAGE SECTION. Independent items are also described in these two sections. For example, a constant that is not logically related to a data organization, such as an interest rate, can be described in the CONSTANT SECTION.

Information such as the size of an item of data, its usage, and its value can be provided in the data-item description. There is one basic format, but not all of the clauses are necessary to describe each item of data. The basic format is:

```
level-number { data-name-1 } [REDEFINES  data-name-2]
             { FILLER      }

[ PICTURE IS { numeric form          } ]
             { alphabetic form       }
             { alphanumeric form     }
             { report form           }
             { scientific decimal form }
             { record mark           } ]

[ SIZE IS positive-integer { CHARACTER [S] } ]
                           { DIGIT [S]     }
```

```
[ CLASS IS  { ALPHABETIC   } ]
            { ALPHANUMERIC }
            { AN           }
            { NUMERIC      }

[ USAGE IS  { COMPUTATIONAL   } ]
            { COMPUTATIONAL-1 }
            { COMPUTATIONAL-2 }
            { DISPLAY         }

[ SYNCHRONIZED { LEFT  } ]        [ SIGNED ]
               { RIGHT }

[ POINT LOCATION IS { LEFT  } ]
                    { RIGHT }
                        positive-integer  PLACE[S] ]

[ BLANK WHEN ZERO ]

[ OCCURS positive-integer TIME[S]
                    [DEPENDING ON data-name-3] ]

[ VALUE IS  { literal            } ]
            { figurative-constant }
```

Each data-item description entry must begin with a level-number and a data-name or the word FILLER. If the REDEFINES clause is used, it must appear immediately after the data-name. The rest of the clauses may appear in any order. The following list gives a summary of the purpose of the data-item description clauses. The rules for each clause are given under the name of the clause.

CLAUSE	USE
level-number	Shows the relation of the data-item to other items.
data-name, FILLER	Gives the name of the data-item.
REDEFINES	Allows alternate grouping of data. Allows one area of storage to be used for two purposes.
PICTURE	Forms a dummy description of the data-item.
SIZE	Specifies the size of the data-item in characters or digits.
CLASS	Tells whether the data-item is numeric, alphabetic, or alphanumeric.
USAGE	Tells whether the data-item is to be used for computations or is to be displayed.
SYNCHRONIZED	Specifies position of a data-item in a computer word.
SIGNED	Specifies a sign.
POINT	Specifies the location of an assumed decimal point.
BLANK	Specifies that the area is to be filled with spaces if the value of the data stored in it is zero.
OCCURS	Allows a data-item description to be repeated.
VALUE	Gives the initial value of a data-item.

Level-Number

A level-number indicates the beginning of a data-item description and shows the relationship of the item to other items.

A level-number may have a value of 01 through 49, or 77 or 88. The level-numbers 01 through 49 are used to show the organization of data-items in logical records. Each field in the logical record is described with a data-item description, and the level numbers show how the fields are related to each other.

The 01 level-number is the most inclusive and must be used for the highest level of data organization, the logical record. Subsequent higher level-numbers indicate the organization of the subdivisions of the logical record. Data-items that are not subdivided are called *elementary* items. Data-items that are subdivided are called *group* items. A group item includes all group and elementary items described under it until a data-item with a level-number numerically less than or equal to the level-number of that group is encountered. Level-numbers need not be assigned consecutively.

The data-items descriptions for a logical record in the FILE SECTION can have the following level-numbers and data-names.

```
01  PERSONNEL-RECORD . . .          group item
    02  AGE . . .                   elementary item
    02  BIRTHPLACE . . .            group item
        03  STATE . . .             elementary item
        03  CITY . . .              elementary item
    02  RESIDENCE . . .             group item
        04  STREET-NAME . . .       elementary item
        04  STREET-NUMBER . . .     elementary item
```

If this logical record is moved to a work area during the program, identical entries are used in the WORKING-STORAGE SECTION to describe the work area.

The level-number 77 is used only in the WORKING-STORAGE SECTION and CONSTANT SECTION. It is used in these sections to identify *independent* data-items, that is, data-items that are not related to other items.

The level-number 88 is used with condition-names. Condition-names may be used in the FILE SECTION or the WORKING-STORAGE SECTION.

Data-Name, FILLER

This clause is used to specify the name of the data being described or to specify a portion of the logical record to which no reference is made. The format of the clause is:

```
level-number  { data-name }
              { FILLER    }
```

Every data-item description must begin with a level-number followed by a data-name or the word FILLER. A data-name is assigned to identify data-items; it refers to the kind of data, not a particular value. A data-name may assume many values during a program.

The data-name is the defining name of the entry, so it need not be unique if it is qualified when referred to.

A data-name may be qualified by writing either IN or OF after it, followed by the name of the group, record, or file in which it is contained. Thus, the data-name YEAR can be assigned to a data-item in a group named DATE and in a group named BIRTH. A unique reference can be made by specifying YEAR OF BIRTH or YEAR IN DATE. There may be any number of levels of qualification. A data-name may not be assigned to a data-item in a group of the same name so that it would appear to qualify itself. Rules for forming data-names are given in "Appendix A."

The key word FILLER is assigned to data-items that are not referred to in the program.

REDEFINES Clause

It is sometimes necessary to use the same storage area for different items at different times in a program. For example, if two work areas REFUND-WORK-AREA and BILLING-WORK-AREA are both needed in a program, they usually occupy two different areas of storage. If these items are never used at the same time in the program, a REDEFINES clause is used to allow them to occupy the same physical area.

Redefinition of an area does not supersede a previous description. Thus, if B and C are two separate items that share the same storage area; the procedure statements MOVE X TO B and MOVE Y TO C can be executed at any point in the program. In the first case, item B assumes the value of X and takes the form specified by the description of item B. In the second case, the same physical area receives Y according to the description of item C.

The format of the REDEFINES clause is:

```
level-number    data-name-1    [REDEFINES data-name-2]
```

When the REDEFINES clause is used, it must immediately follow *data-name-1,* the defining name of the entry. The entries describing the new area of storage (*data-name-1* and its subdivisions) must immediately follow the entries describing the area being redefined (*data-name-2* and its subdivisions).

Redefinition starts at *data-name-2* and ends when a level-number less than or equal to that of *data-name-2* is encountered. The storage area for *data-name-1* must not be larger than that for *data-name-2* and the level-numbers must be identical. *Data-name-2* should be qualified when necessary, but subscripting is not permitted. (Subscripting and qualification are explained in "Appendix A.")

The REDEFINES clause must not be used in the FILE SECTION with logical records (01 level) associated with the same file since the DATA RECORDS clause in the file description entry implies automatic redefinition. The entries giving the new description of the storage area must not contain VALUE clauses assigning initial values to the area; however, the VALUE clause may be used for condition-name entries.

The following entries in the WORKING-STORAGE SECTION redefines REFUND-WORK-AREA so that it can be used for another purpose in a different part of the program. The redefinition includes everything under BILLING-WORK-AREA until either a 01 level-number or the end of the WORKING-STORAGE SECTION occurs.

```
01  REFUND-WORK-AREA.
      .
      .
01  BILLING-WORK-AREA REDEFINES
                              REFUND-WORK-AREA.
      .
      .
```

Another use of the REDEFINES clause is given in "Arrays and Subscripts" in Part II of this publication.

PICTURE Clause

The PICTURE clause provides a compact form for specifying the characteristics of an *elementary* data-item. Each character position in the data-item is represented by a code, and the combination of the codes form a dummy or blueprint of the data-item. Instead of repeating one code several times, the code may be written once followed by a number in parentheses indicating the number of times the code is repeated. For example, the PICTURE 999999 can be written 9(6). There may be a maximum of 30 characters in the code. The PICTURE clause may be used (and should be for good practice) instead of the SIZE, CLASS, SIGNED, and POINT clauses. The format of the PICTURE clause is:

```
[ PICTURE IS { numeric form
             { alphabetic form
             { alphanumeric form
             { report form
             { scientific decimal form
             { record mark                 } ]
```

Numeric Form: The numeric form is used for numeric data-items, that is, those composed of the numbers 0 through 9 and an operational sign. The maximum number of characters allowed in numeric data-items is 18. The PICTURE may contain a combination of codes 9 V P S.

Code 9

The code 9 indicates that the character position always contains a numeric character. The number 123 could be a value of a data-item with the PICTURE 999.

Code V

The code V indicates the position of an *assumed* decimal point. Numeric items cannot contain actual decimal points. No storage is reserved for

the decimal point, so it is not counted in determining the SIZE of the item. It informs the compiler of the alignment of elementary items used for computations. The number 1.23 could be a value of a data-item with the PICTURE 9V99.

Code P

The code P represents a numeric character position for which storage is not reserved. It is used to indicate the location of an assumed decimal point and is treated as though it contained a zero. The code V may be omitted. The number .00123 can be the value of a data-item with the PICTURE VPP999, PP999, or P(2)9(3). The number 12300 can be the value of a data-item with the PICTURE 999PPV, 999PP, or 9(3)P(2). The SIZE of both of these items is 3.

Code S

The code S indicates the presence of an operational sign. If the USAGE of the item is COMPUTATIONAL, it is redundant to specify the character S, since computational items are assumed to be signed; if the USAGE is DISPLAY, S specifies a sign over-punch in the units position. If used, S must always be written as the leftmost character of the PICTURE. The number −123000 can be the value of a data-item with the PICTURE S999999 or S9(6).

Alphabetic Form: An alphabetic data-item may contain only the letters of the alphabet and the space. Alphabetic data is required to be USAGE DISPLAY. The PICTURE of an alphabetic data-item may contain only the code A. The PICTURE of a data-item containing the word *output* can be AAAAAA or A(6).

Alphanumeric Form: An alphanumeric data-item may contain any character in the COBOL character set. Alphanumeric data is required to be USAGE DISPLAY. The PICTURE of an alphanumeric data-item may contain the codes 9 A X and must contain at least one A or X. The codes 9 (numeric position) and A (alphabetic position) have been explained. An X indicates that the character position may contain any character in the COBOL character set. A mixture of the codes is treated as though the PICTURE were all X's. The PICTURE of the word Type 2 could be XXXXX or X(5).

Report Form: Data editing is accomplished by moving the source data to a data-item that has the edited or report form of the PICTURE clause. Computations are performed before the data is edited since edited items are ALPHANUMERIC DISPLAY and cannot be used for computations. The editing specifies the insertion, replacement, and/or suppression of certain characters. The PICTURE of an edited data-item may contain a combination of the codes:

9 V P Z * . , 0 B CR DB + − $

The codes 9 V P have the same meaning as in the numeric form.

Suppression Characters

Code Z

The code Z specifies that the character position is to be replaced by a space if a nonsignificant leading zero appears in this position in the source item. The Z must never be preceded by a 9.

SOURCE ITEM PICTURE	SOURCE ITEM	EDITING PICTURE	EDITED ITEM
9999	0102	ZZZZ	102
9999	0012	Z999	012

Code *

The asterisk (*) indicates check protection, that is, the replacement of nonsignificant zeros by asterisks. An asterisk must never be preceded by a 9.

SOURCE ITEM PICTURE	SOURCE ITEM	EDITING PICTURE	EDITED ITEM
9999	0000	****	****
9999	0030	****	**30
99999	00100	***99	**100
99999	00001	***99	***01

Replacement Characters – A place must be reserved for replacement characters, so the PICTURE must be one character longer than the greatest number of characters that are to be moved into it.

Code Floating + − $

Zero suppression by means of a floating + or − or $ is specified by placing a string of plus signs, minus signs, or dollar signs in the numeric character positions to be suppressed. The floating character used appears immediately to the left of the first significant digit of the source item.

If the value of the source item is negative, a floating plus or minus sign causes a minus to be placed in the edited item. If the value of the source item is positive or zero, a floating plus sign causes a plus to be inserted, and a floating minus sign causes a space to be inserted in the edited item. The string of floating characters must be the leftmost characters in a PICTURE. The insertion characters comma, B, and 0 may be embedded, noncontiguously, in a string. Leading commas, B's, and 0's are suppressed.

SOURCE ITEM PICTURE	SOURCE ITEM	EDITING PICTURE	EDITED ITEM
9999	0315	$$99	$315
99V99	3456	$$,$$$.99	$34.56
99999	00315	+++99	+315
99999	00315	−−999	315

Insertion Characters – A place must be reserved for the insertion character so one character code must be added to the PICTURE for each insertion character.

Code .

The decimal point (.) specifies that an *actual* decimal point is to be inserted in the indicated position and the source item is to be aligned accordingly. Numeric positions to the right of the decimal point in the PICTURE must all have the same code. Unlike the assumed decimal point, the actual decimal point occupies a character position and is counted in determining the SIZE of an item.

SOURCE ITEM PICTURE	SOURCE ITEM	EDITING PICTURE	EDITED ITEM
99V99	6321	99.99	63.21
V9999	0311	.9999	.0311

Code , 0 B

The comma, zero, and B specify the insertion of comma, zero, and space, respectively. If all characters to the left of the insertion character have been suppressed, the insertion character is suppressed. If a string of identical insertion characters is specified, it is replaced by a single character, and a warning message is issued. Contiguous insertion characters of different types cause an error that prevents execution of the program. A zero to the right of a decimal point is treated as the numeric code 9.

SOURCE ITEM PICTURE	SOURCE ITEM	EDITING PICTURE	EDITED ITEM
999999	125739	99B99B99	12 57 39
999999	123456	9990999	1230456
9999V99	123412	9,999.99	1,234.12
99999V99	412	ZZ,ZZZ.99	4.12

Code CR DB

CR and DB are credit and debit symbols. The symbol may appear only as the *two* rightmost characters of a PICTURE and appears in the indicated position if the source item is negative. If the value of the source item is positive or zero, spaces appear in the edited item.

SOURCE ITEM PICTURE	SOURCE ITEM	EDITING PICTURE	EDITED ITEM
99V99	−6325	$99.99CR	$63.25CR
99V99	6325	$99.99CR	$63.25

Code + − $

The single plus, minus, and dollar sign specify the insertion of + or − or $. If the value of the source item is negative, a single plus or minus sign causes a minus to be placed in the indicated position. If the value of the source item is positive or zero, a single plus sign causes a plus to be inserted, and a single minus sign causes a space to be inserted. The dollar sign causes a dollar sign to be placed in the indicated position.

The single dollar sign must be the first code in the PICTURE. The single plus or minus sign may be written as either the first or the last code in the PICTURE.

SOURCE ITEM PICTURE	SOURCE ITEM	EDITING PICTURE	EDITED ITEM
9999	0315	$Z999	$ 315
9999	−0411	+9999	−0411
9999	0321	−ZZ99	321

Scientific Decimal Form: The scientific decimal form of a PICTURE clause specifies a special type of editing. The PICTURE of a scientific decimal item may contain the codes:

+ − 9 . V E

A scientific decimal number has two components, the mantissa and the exponent. The mantissa is a signed or unsigned decimal number with or without a decimal point. The exponent is a signed or unsigned decimal integer. It is the power of ten by which the mantissa is multiplied.

In the PICTURE, the mantissa must contain a sign and one to sixteen 9's and may contain a decimal point represented by a . or V. The exponent is preceded by the letter E and must contain a sign and two 9's. Specifically the PICTURE must have the form:

$$\underbrace{\left\{\pm\right\}\left[9(m)\right]\left[\left\{V\right\}\right]\left[9(n)\right]}_{\text{mantissa}}\quad\underbrace{E\left\{\pm\right\}99}_{\text{exponent}}$$

where

m and n are positive integers

m + n must be greater than zero and less than 17. The symbol E is used to separate the exponent from the mantissa. There must be no embedded blanks in the PICTURE.

SOURCE ITEM PICTURE	SOURCE ITEM	EDITING PICTURE	EDITED ITEM
9999999	3000000	9E+6	3×10^{6}
V9999	.0012	9.9E−3	1.2×10^{-3}

Record Mark: The code J specifies that the single character record mark is to appear as a constant. If J is used, the PICTURE must contain only one J and no other character may appear with it. The record mark is used with peripheral equipment.

SIZE Clause

The SIZE clause specifies the number of alphanumeric characters or numeric digits in the data-item. The size of elementary data-items must be specified by a PICTURE or a SIZE clause.

Use of the SIZE clause for a group item has no effect on the program. The format of the SIZE clause is:

```
[ SIZE  IS positive-integer  { CHARACTER[S] } ]
                             { DIGIT[S]     }
```

The words CHARACTERS and DIGITS are equivalent in the format. Operational signs (as indicated by a SIGNED clause or by an S in the PICTURE clause), assumed decimal points (as specified by a V in the PICTURE clause or by a POINT clause), or assumed character positions (as indicated by a POINT clause or by a P in the PICTURE clause) should not be counted in determining the size of an item.

If there is conflict between the PICTURE and SIZE clauses for a data-item, PICTURE dominates.

CLASS Clause

The CLASS clause indicates whether the data-item is alphabetic, numeric, or alphanumeric. If a CLASS clause is given for a group item, it applies to each elementary item in the group. The format of the CLASS clause is:

```
[ CLASS IS { ALPHABETIC  } ]
           { NUMERIC     }
           { ALPHANUMERIC }
           { AN          }
```

An ALPHABETIC data-item consists of any combination of the 26 characters of the alphabet and a space.

A numeric data-item consists of any combination of the numbers 0 through 9 and an operational sign. Unsigned data is assumed to be positive. Data-items whose USAGE IS COMPUTATIONAL are assumed to be NUMERIC; for such items, this clause may be omitted.

An ALPHANUMERIC data-item may contain any character in the COBOL character set; therefore, ALPHABETIC and NUMERIC items within an ALPHANUMERIC group are not considered contradictory. The abbreviation for ALPHANUMERIC is AN.

If there is a conflict between the PICTURE and CLASS clauses for a data-item, PICTURE dominates.

USAGE Clause

The USAGE clause describes the representation of data in core storage. Data may be stored in either BCD (Binary Coded Decimal) or binary form. In BCD form, each character in the COBOL character set is represented by a code. The octal representation of the BCD character code is given in "Appendix B." The binary form of a number is the representation of the number to the base 2.

The format of the USAGE clause is:

```
[ USAGE IS { COMPUTATIONAL   } ]
           { COMPUTATIONAL-1 }
           { COMPUTATIONAL-2 }
           { DISPLAY         }
```

The word DISPLAY means that the item is stored in BCD form. If USAGE is not specified for an elementary item or in the description of a group to which it belongs, the USAGE is assumed to be DISPLAY. DISPLAY must not be specified for items in files on magnetic tape recorded in binary mode. The USAGE clause is not required for ALPHANUMERIC data since it is assumed to be DISPLAY.

The word COMPUTATIONAL means that the item is stored in binary form and is to be used for computations.

The three types of COMPUTATIONAL items are:

COMPUTATIONAL	double-precision floating-point
COMPUTATIONAL-1	single-precision floating-point
COMPUTATIONAL-2	double-precision floating-point

The word COMPUTATIONAL is used to describe a real number to the base 10 that is used for computations.

The suffixes (−1 and −2) are used to describe floating-point items. A floating-point number has the form $a * 10^b$ where a, the mantissa, and b, the exponent, are real numbers to the base 10.

COMPUTATIONAL-1 describes single-precision floating-point numbers; that is, those contained in one computer word. The mantissa may have from 1 to 8 digits and the exponent may have 1 or 2 digits. COMPUTATIONAL-2 describes double-precision floating-point numbers; that is, those contained in two computer words. The mantissa may have from 1 to 16 digits and the exponent may have 1 or 2 digits. For both single- and double-precision floating-point numbers, the exponent may range from −38 to +38. The USAGE clause is the only clause necessary for floating-point data.

The COMPUTATIONAL items are assumed to be NUMERIC, so the CLASS need not be specified. COMPUTATIONAL must not be specified for items in files on magnetic tapes recorded in BCD mode.

Combined SIZE, CLASS, and USAGE Clause

The SIZE clause may be combined with the CLASS clause and/or the USAGE clause. The rules for the combined clause are the same as those for the individual clauses. The order in which CLASS and USAGE are specified is not important. The format is:

```
[ SIZE IS integer [ { ALPHABETIC  } ] ]
                  { NUMERIC     }
                  { ALPHANUMERIC }
                  { AN          }

  [ { COMPUTATIONAL   } ] { CHARACTER[S] } ]
  [ { COMPUTATIONAL-1 } ] { DIGIT[S]     }
  [ { COMPUTATIONAL-2 } ]
  [ { DISPLAY         } ]
```

SYNCHRONIZED Clause

The SYNCHRONIZED clause specifies the positioning of an elementary item in a computer word or words. The format is:

```
[ SYNCHRONIZED { LEFT  } ]
               { RIGHT }
```

SYNCHRONIZED LEFT indicates that the leftmost character of the data-item is to occupy the left-hand portion of the next computer word. This may mean that the right-hand portion of the preceding word is unoccupied.

SYNCHRONIZED RIGHT indicates that the rightmost character of the data-item is to occupy the right-hand portion of the next computer word. This means that the data-item apears in a word (two words, if more than 10 digits) by itself. The most efficient form for NUMERIC COMPUTATIONAL items is SYNCHRONIZED RIGHT, since the item can be used for computation as it is. An unsynchronized computational item or one that is SYNCHRONIZD LEFT may have to be put into a computer word by itself each time it is used.

SIGNED Clause

The SIGNED clause is used at the elementary level to specify a standard operational sign. The sign may also be shown by an S in the PICTURE clause. The format is:

```
[ SIGNED ]
```

The SIGNED clause is most often used with DISPLAY items. The standard operational sign for DISPLAY items is in the form of an overpunch in the rightmost position.

The standard operational sign for COMPUTATIONAL items is in the leftmost position. A COMPUTATIONAL item is always signed; therefore, it is redundant to specify an operational sign.

An item containing an operational sign must be numeric; therefore, a CLASS clause for the item need not be given. The operational sign is not considered in determining the size of an item. If both a PICTURE clause and a SIGNED clause are used, the PICTURE must not contain editing symbols.

POINT LOCATION Clause

The POINT LOCATION clause specifies the position of an *assumed* decimal point for elementary data-items. The location of an assumed decimal point may also be given by a V in the PICTURE clause. The assumed decimal point determines the alignment of data for computations. An *actual* decimal point must be given by a decimal point (.) in the report form of the PICTURE clause. The format of the POINT LOCATION clause is:

```
[ POINT LOCATION IS { LEFT  }
                    { RIGHT }
                                positive-integer PLACE[S] ]
```

If the location of a decimal point is not given, or implied by a P in the PICTURE clause, the item is assumed to be an integer; that is, the decimal point is at the right-hand end of the data-item. In the format, *positive-integer,* gives the location of the decimal point as the number of character positions to the left or right of the right-hand end of the data-item. For example, if a data-item is to contain an interest rate that may vary from .03 to .08 the data-item description can be:

```
06 INTEREST-RATE SIZE IS 2 COMPUTATIONAL
   CHARACTERS, SYNCHRONIZED RIGHT, POINT
   LOCATION IS LEFT 2 PLACES.
```

If the numbers vary from 3000 to 8000, the clause, POINT LOCATION IS RIGHT 2 PLACES, can be substituted.

BLANK WHEN ZERO Clause

This clause specifies that the item is filled with blanks (spaces) whenever the value of the item is zero. The format is:

```
[ BLANK WHEN ZERO ]
```

The BLANK WHEN ZERO clause may be used only at the elementary level. It must not be used if all the numeric character positions in the PICTURE clause contain asterisks (*), but it overrides all other editing specifications in a PICTURE clause.

Since a BLANK WHEN ZERO clause specifies editing, the data-item is considered to be USAGE DISPLAY and CLASS ALPHANUMERIC.

OCCURS Clause

The OCCURS clause is used to repeat a data-item description. The number of times the description is to be repeated may be given directly in the OCCURS clause, or it may be calculated when the program is executed.

Since the repeated items do not have individual defining data-names, they must be subscripted when referred to. For example, if a data-item TABLE is described as OCCURS 10 TIMES, the third occurrence is referred to as TABLE(3). Any subdivision of a repeated item must also be subscripted when referred to. The rules for using subscripts are given in "Appendix A."

The format of the OCCURS clause is:

```
[ OCCURS positive-integer TIME[S]
                        [DEPENDING ON data-name] ]
```

When the DEPENDING ON option is not used, *positive-integer* is the number of occurrences of the data-item.

The DEPENDING ON option permits the number of repetitions of the data-item description to be determined during the execution of the program. When the data-item is referred to, the number of repetitions is set equal to the current value of *data-name*. Thus, the number of repetitions of the data-item description may vary during the program. When this option is used, *positive-integer* is the maximum number of repetitions and is used for storage reservation.

Data-name must be an elementary data-item with an integral value greater than zero; or it may be the special register TALLY. It may be qualified when used, but subscripting is not permitted. *Data-name* must be described prior to any variable-length portion of a data organization. For example, if *data-name* and the data-item containing the OCCURS clause are subdivisions of the same logical record, *data-name* must be described first.

The OCCURS clause may not be used at the 01 level.

VALUE Clause

The VALUE clause specifies the value of an elementary data-item. In the FILE SECTION, the VALUE clause may be used only in condition-name entries; in the CONSTANT SECTION, it may be used only to specify the value of a constant; and in the WORKING-STORAGE SECTION, it may be used for either purpose.

The format of the VALUE clause is:

```
[VALUE IS {literal              }]
          {figurative-constant  }
```

A *literal* is a word or number that is a constant. It has a fixed value that never changes during the execution of a program. There are two classes of literals: numeric and nonnumeric. For example, both of the following are literals:

```
     .10
"THIS IS A NONNUMERIC LITERAL"
```

Numeric literals may contain from 1 to 18 characters chosen from the following: the numbers 0 through 9, a plus or a minus sign, and a decimal point. There may be, at most, one sign in the literal and it must appear in the leftmost position. If a decimal point is not present, it is assumed to be in the rightmost position.

Floating-point literals are a special type of numeric literal and are written in floating-point literal format:

```
[{±}] mantissa    E [{±}] exponent
```

The mantissa may contain 1 to 16 digits and a decimal point (except in the rightmost position). The exponent may contain one or two digits whose magnitude may not exceed 38. There may be no embedded blanks in the literal. Floating-point literals may be used only in the DATA DIVISION, but other numeric and nonnumeric literals may also be used in the PROCEDURE DIVISION.

Numeric literals must not be enclosed in quotation marks. Nonnumeric literals must be enclosed in quotation marks. If a literal conforms to the rules for the formation of numeric literals, but is enclosed in quotation marks, it is assumed to be a nonnumeric literal.

Nonnumeric literals may be composed of any characters in the COBOL character set except the quotation mark and may contain up to 120 characters. All spaces enclosed in the quotation marks are included in the size of the literal.

A *figurative constant* is a value that has a preassigned fixed data-name. It is not enclosed in quotation marks. The fixed data-names and their meanings are given below. The singular and plural forms may be used interchangeably.

ZERO ZEROS ZEROES	Represent one or more zeros (0).
SPACE SPACES	Represent one or more blanks or spaces.
HIGH-VALUE HIGH-VALUES	Usually represent one or more 9's, the highest value in the commercial collating sequence; but represent one or more left parentheses if the scientific (binary) collating sequence has been specified by the option BINSEQ on the $IBCBC control card.
LOW-VALUE LOW-VALUES	Usually represent one or more blanks or spaces, the lowest value in the commercial collating sequence; but represent one or more zeros if the scientific (binary) collating sequence has been specified by the option BINSEQ on the $IBCBC control card.
QUOTE QUOTES	Represent the character '. Note that the use of the figurative constant QUOTE to represent the character ' is not equivalent to the use of the symbol ' to bound a literal.
ALL 'literal'	Represents one or more occurrences of *'literal'*. *'Literal'* represents a single character nonnumeric literal and must be enclosed in quotation marks. An alternative form of *'literal'* is any figurative constant except ALL, e.g., ALL SPACES.

The only figurative constant that may be used with a numeric data-item is ZERO (ZEROS, ZEROES). All of the figurative constants may be used with ALPHANUMERIC or ALPHABETIC data-items.

The VALUE clause may not be used with a data-item description that contains an OCCURS clause, or with a data-item that is a subdivision of an entry containing an OCCURS clause, or with a data-item that is described after an entry in the same logical record that contains an OCCURS DEPENDING ON clause.

File Section

The FILE SECTION contains a description of all externally stored data. Externally stored data is organized into files made up of logical records. The two types of entries in the FILE SECTION are: *file description* entries and *data-item description* entries. The file description entries appear only in the FILE SECTION. They describe the physical characteristics of the file. The data-item descriptions in the FILE SECTION describe the logical records on the file.

File Description Entry

A file description entry must precede the entries describing the data in the file. The information provided in this entry (e.g., the recording mode of the file, label information, and the names of the records on the file) is necessary for reading the data in the file into core storage or writing data from core storage.

The format of the file description entry is:

```
FD file-name

DATA {RECORD IS  } record-name-1 [record-name-2 . . .]
     {RECORDS ARE}

[RECORDING MODE IS {BCD   } {LOW }
                   {BINARY} {HIGH}
            DENSITY [WITHOUT COUNT CONTROL]]

[BLOCK CONTAINS [integer-1 TO] integer-2
                               {RECORD[S]   }]
                               {CHARACTER[S]}
[RECORD CONTAINS [integer-1 TO] integer-2
                                CHARACTERS]

LABEL {RECORDS ARE} {STANDARD}
      {RECORD IS  } {OMITTED }

[VALUE OF label-data-name-1 IS literal-1
                    [label-data-name-2 IS . . .]]
```

Each file description entry must begin with FD and the name of the file. The rest of the clauses may be in any order. The following list summarizes the purpose of the file description clauses:

CLAUSE	USE
FD file-name	Identifies the beginning of a file description entry.
DATA RECORDS	Names the records on the file.
RECORDING MODE	Specifies the recording mode, density, and presence or absence of count control words.
BLOCK CONTAINS	Gives the number of records or characters in a block.
RECORD CONTAINS	Gives the number of characters in the records.
LABEL	Tells whether the label is standard, nonstandard, or omitted.
VALUE	Gives values of fields in the standard label.

Figure 7 is a sample file description entry. The rules for using the file description clauses are given under the names of the individual clauses.

FD, File-name

Each file description entry must begin with FD and the name of the file. The level indicator FD identifies the beginning of a file description. The file-name is the name the programmer assigns to the file. This is the name given in the FILE-CONTROL paragraph in the ENVIRONMENT DIVISION. The format of this clause is:

```
FD file-name
```

File-name is formed according to the rules for names (given in "Appendix A") and must not be qualified.

DATA RECORDS Clause

This required clause provides a cross reference between logical records and their associated file by naming the logical records on the file. The format is:

```
DATA {RECORD IS  } record-name-1 [record-name-2 . . .]
     {RECORDS ARE}
```

Record-name-1, record-name-2, etc., are the names of the logical records on the file. The presence of more than one record-name indicates that the file may contain records of different sizes and/or formats. The order in which they are specified is not significant.

```
170     ALL FILES.
180 DATA DIVISION.
190 FILE SECTION.
200 FD  EMPLOYEE-FILE  RECORDING MODE IS BINARY
210     BLOCK CONTAINS 6 RECORDS  LABEL RECORDS ARE
220         STANDARD
230     VALUE OF FILE-IDENTIFICATION IS 'EMPLOYEES'  VALUE OF
240         FILE-SERIAL-NUMBER IS '213A'
250     DATA RECORDS ARE PAYROLL, PERSONAL.
```

*A standard card form, IBM electro C61897, is available for punching source statements from this form.

Figure 7. Sample File Description Entry

The logical records, *record-name-1, record-name-2,* are described by data-item descriptions with 01 level-numbers. Subscripting of these names is not permitted and qualification is not necessary since they are implicitly qualified by the file-name of the entry. Subscripting and qualification are explained in "Appendix A."

RECORDING MODE Clause

The RECORDING MODE clause describes the form of the data in the file. It indicates the recording mode and density of the file. For files containing records of different lengths, it specifies whether there is a control word preceding each record to indicate the size of the record. The format of the clause is:

```
[RECORDING MODE IS {BCD   }[{LOW  }DENSITY]
                   {BINARY}[{HIGH }       ]
                            [WITHOUT COUNT CONTROL]
```

Tapes recorded in binary mode contain data in either binary or BCD code. Tapes recorded in BCD mode contain only BCD information; therefore, data on BCD tapes must be described (explicitly or implicitly) in data-item descriptions as USAGE IS DISPLAY. The recording mode of the file must agree with the recording mode of the unit to which it is assigned if there is a system requirement. The BCD mode should be specified for any file assigned exclusively to card equipment.

Either HIGH DENSITY or LOW DENSITY may be specified. The density is the number of characters written on a given area of tape; the more characters, the higher the density. Some auxiliary equipment requires the information to be recorded in low density. However, if the tape is to be completely processed on the IBM 7090/7094, the HIGH DENSITY option should be used to minimize the tape area required to contain a given number of characters.

Usually, if a file assigned to a tape unit contains different length records, each record is preceded by a control word showing the number of words in the record. (These words should be counted in determining the number of characters in the record if the CHARACTERS form of the BLOCK CONTAINS clause is used.) If WITHOUT COUNT CONTROL is specified, these control words are not expected when the tape is read, nor are they supplied when the tape is written.

If this clause is omitted, RECORDING MODE IS BCD HIGH DENSITY is assumed.

BLOCK CONTAINS Clause

Several logical records may be blocked when they are stored on magnetic tape in order to save time in transmitting the data and to conserve space on the tape. A physical record is a block of one or more logical records. Physical records are separated on magnetic tape by interrecord gaps. Figure 8 shows logical records blocked on magnetic tape.

The BLOCK CONTAINS clause specifies the size of the blocks or physical records in the file. The format of the BLOCK CONTAINS clause is:

```
BLOCK CONTAINS [integer-1 TO] integer-2
                              {RECORD[S]   }
                              {CHARACTER[S]}
```

A logical record on a file may be either *fixed* or *variable* in length. A fixed-length record is one of predetermined size. A variable-length record is one whose size is not determined until the program is run; that is, one that contains an OCCURS DEPENDING ON clause in the data-item description entries. A file may contain more than one type of logical record; therefore, a combination of fixed- and/or variable-length records may be on the file.

The RECORDS form of the BLOCK CONTAINS clause is used to specify the size of the block by the number of logical records in the block. This is usually used for files containing one or more types of fixed-length records. The following rules apply to the RECORDS form of the BLOCK CONTAINS clause:

1. *Integer-1* and *integer-2* are the minimum and maximum number of logical records in the block.
2. If only *integer-2* is specified, each block on an input file must contain *integer-2* records and exactly *integer-2* records are written in a block for an output file.

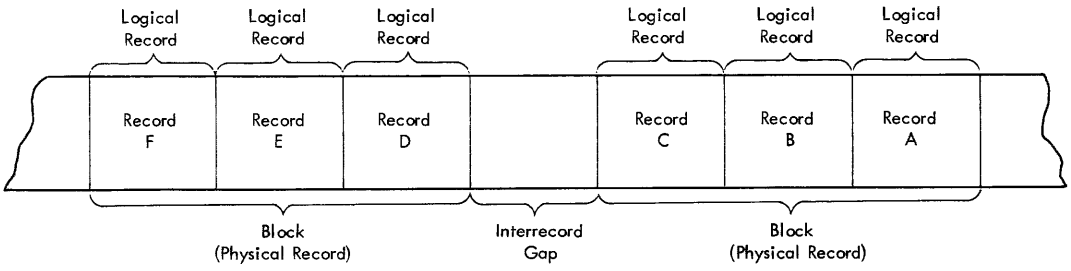


Figure 8. Logical Records in Blocks on Magnetic Tape

3. If *integer-1* TO *integer-2* is specified, each block on an input file must contain from *integer-1* to *integer-2* records, and *integer-2* records are written in a block for an output file.

The CHARACTERS form of the clause specifies the size of the block by the number of computer characters (six computer characters equal one computer word) in the block. This form may be used for files containing variable-length records. The following rules apply to the use of the CHARACTERS form of the BLOCK CONTAINS clause:

1. *Integer-1* and *integer-2* are the minimum and maximum number of characters in the block.
2. *Integer-1* and *integer-2* must be evenly divisible by six so that there are an integral number of computer words in the block. If they are not evenly divisible by six, a warning message is issued; on an output file this may mean that the rightmost positions of a record contain unwanted information.
3. Each block contains an integral number of logical records.
4. If only *integer-2* is specified, each block on an input file must contain up to *integer-2* characters and each block on an output file contains the greatest number of logical records possible without exceeding *integer-2* characters.
5. If *integer-1* TO *integer-2* is specified, each block on an input file must contain from *integer-1* to *integer-2* characters and each block on an output file contains the greatest number of logical records possible without exceeding *integer-2* characters.

Care should be taken when using the CHARACTERS form of the BLOCK CONTAINS clause since it applies to the number of characters that are reserved for the block in core storage, not to the number of characters in the record.

1. The use of the SYNCHRONIZED clause in the data-item description affects the number of characters reserved in storage. Figure 9 shows the number of characters that are reserved for synchronized items.
2. Control words preceding the block must be counted in determining the block size. These words are explained in the RECORDING MODE clause.
3. If the APPLY clause of the I-O-CONTROL paragraph in the ENVIRONMENT DIVISION has not been used and there are block sequence words appended to the blocks, one extra word (six characters) must be added to the block size.

The BLOCK CONTAINS clause is used to determine buffer sizes. A buffer area is an area reserved in core storage to hold a physical record before it is needed in a program or after it has been processed by a program. The input-output system automatically reads physical records into buffer areas and writes physical records out of buffer areas. The programmer is concerned only with logical records. When a READ instruction is given,

Number of characters or digits in Size or Picture Clauses	Number of Characters Reserved in Core Storage			
	Computational Items		Display Items	
	Synchronized Right (1)	Synchronized Left or Unsynchronized (2)	Synchronized Right (1)	Synchronized Left or Unsynchronized (2)
1	6	1	6	1
2	6	2	6	2
3	6	2	6	3
4	6	3	6	4
5	6	3	6	5
6	6	4	6	6
7	6	5	12	7
8	6	5	12	8
9	6	6	12	9
10	6	6	12	10
11	12	7	12	11
12	12	7	12	12
13	12	8	18	13
14	12	9	18	14
15	12	9	18	15
16	12	10	18	16
17	12	10	18	17
18	12	11	18	18

(1) Items that are synchronized right are stored in computer words by themselves. This may mean that the rightmost portion of the preceding computer word will be unoccupied. This unoccupied portion of the word must also be counted for storage reservation.

(2) Items that are synchronized left are stored in the leftmost portion of the next available computer word. This may mean that the rightmost portion of the preceding word is empty. This unoccupied portion of the word must also be counted for storage reservation. An item that follows a synchronized-left item is stored in the same word with it if there is room (unless, of course, the second item is also synchronized either right or left).

Figure 9. Effect of Synchronization on Allocation of Core Storage

one logical record is located in the buffer area. When a WRITE instruction is given, one logical record is released to be written out of core storage from the buffer area. Figure 10 shows how the BLOCK CONTAINS clause determines the size of the buffer areas. It also contains a summary of the interpretation of the four permissible forms of the BLOCK CONTAINS clause. Blocks should contain more than three computer words in order to insure proper transmission of data.

The BLOCK CONTAINS clause is not required if each physical record in the file contains only one logical record. The form BLOCK CONTAINS 1 RECORD is assumed if the clause is omitted. If the word RECORDS is not written in the clause, CHARACTERS is assumed.

Figure 11 gives a summary of descriptions that can be provided for different types of files. It shows the relationship of the FILE-CONTROL paragraph in the ENVIRONMENT DIVISION and the DATA RECORDS, the RECORDING MODE, and the BLOCK CONTAINS clauses in the file description entry.

RECORD CONTAINS Clause

The RECORD CONTAINS clause specifies the number (or range) of characters in the logical records of a file. This is an optional clause that is used primarily for checking the record size specified in the data-item descriptions. The format is:

```
RECORD CONTAINS [integer-1 TO] integer-2
                                CHARACTERS
```

Integer-1 and *integer-2* are positive integers that specify the number of characters in the smallest record and the number of characters in the largest logical record, respectively. If only *integer-2* is specified, all logical records in the file are assumed to be the same length.

In this clause CHARACTERS is the number of computer characters (six characters in a computer word) in the logical records. When specifying the number of characters in the records, synchronization of computational data-items must be taken into consideration. Figure 9 specifies the relationship.

LABEL RECORDS Clause

A label is a record or block of records that identifies a file. The LABEL RECORDS clause is required and indicates the presence or absence of standard labels. A standard label has a fixed format. Since the subdivisions of the standard label have been assigned data-names, the programmer can specify the values

Form of Clause	Record Types on the File	Size of Buffer Area (1)	Interpretation of the Clause (2)
integer-2 RECORDS	Records of one type; i.e., one record name in the DATA RECORDS clause.	integer-2 * record-length	Each block on an input file must contain exactly integer-2 logical records. Exactly integer-2 logical records will be written in a block for an output file.
	Records of more than one type; i.e., more than one record name in the DATA RECORDS clause.	integer-2 * length of the longest record	Same interpretation as for records of one type.
integer-1 TO integer-2 RECORDS	Records of one type.	integer-2 * record-length	Each block on an input file may contain from integer-1 to integer-2 logical records. Integer-2 logical records will be written in a block for output files.
	Records of more than one type.	integer-2 * length of the longest record	Some interpretation as for records of one type.
integer-2 CHARACTERS	Records of one type.	integer-2 / 6	Each block on both input and output files contains integer-2 characters.
	Records of more than one type.	integer-2 / 6	Each block on an input file may contain up to integer-2 character; each block on an output file as many logical records as possible without exceeding integer-2 characters.
integer-1 TO integer-2 CHARACTERS	Records of one type.	integer-2 / 6	Each block on an input file may contain from integer-1 to integer-2 characters. Each block on an output file will contain as many logical records as possible without exceeding integer-2 characters.
	Records of more than one type.	integer-2 / 6	Same interpretation as for records of one type.

(1) The asterisk (*) means multiplication and the slash (/) means division. Record-length is the number of computer words in the logical record.

(2) The last block on the file will not necessarily be filled. When a block is not filled, it is truncated, not padded.

Figure 10. Determination of Buffer Size

Type	Use of File		Blocked	Block Specification		Unit Type		
	Input	Output		Records	Characters	Tape	Card	System Unit
1	yes	no	no	—	—	yes	no	no
2	yes	no	no	—	—	no	yes	no
3	yes	no	no	—	—	no	no	yes (1)
4	no	yes	no	—	—	yes	no	no
5	no	yes	no	—	—	no	yes	no
6	no	yes	no	—	—	no	no	yes (2)
7	yes	no	no	—	—	yes	no	no
8	yes	no	no	—	—	no	yes	no
9	yes	no	no	—	—	no	no	yes (1)
10	no	yes	no	—	—	yes	no	no
11	no	yes	no	—	—	no	yes	no
12	no	yes	no	—	—	no	no	yes (2)
13	yes	no	yes	yes	yes	yes	no	no
14	no	yes	yes	yes	yes	yes	no	no
15	yes	no	yes	yes	yes	yes	no	no
16	no	yes	yes	no	yes	yes	no	no
17	no	yes	yes	yes	no	yes	no	no
18	yes	no	yes	yes	yes	yes	no	no
19	no	yes	yes	no	yes	yes	no	no
20	no	yes	yes	yes	no	yes	no	no

Type	Recording Mode		Control Words	Type of Record (3)		Length of Record	
	BCD	Binary		One	More than one	Fixed	Variable
1	yes	yes	no	yes	no	yes	no
2	yes	no	no	yes	no	yes	no
3	yes	yes (4)	no	yes	no	yes	no
4	yes	yes	no	yes	no	yes	no
5	yes	no	no	yes	no	yes	no
6	yes	no	no	yes	no	yes	no
7	yes	yes	optional	no	yes	yes	yes
8	yes	no	no	no	yes	yes	yes
9	yes	yes (4)	optional (5)	no	yes	yes	yes
10	yes	yes	optional	no	yes	yes	yes
11	yes	no	no	no	yes	yes	yes
12	yes	no	optional (5)	no	yes	yes	yes
13	yes	yes	no	yes	no	yes	no
14	yes	yes	no	yes	no	yes	no
15	yes	yes	optional	yes	no	no	yes
16	yes	yes	optional	yes	no	no	yes
17	yes	yes	optional	yes	no	no	yes
18	yes	yes	yes	no	yes	yes	yes
19	yes	yes	yes	no	yes	yes	yes
20	yes	yes	yes	no	yes	yes	yes

(1) May be SYSIN1 only.

(2) May be SYSOU1 or SYSPP1 only.

(3) "One type record" means one record-name in DATA RECORDS clause; "more than one type" means more than one record-name in DATA RECORDS clause.

(4) Binary recording mode permissible only if unit medium is tape.

(5) Control word permissible only if unit medium is tape.

Figure 11. File Characteristics

of these names with the VALUE clause in the file description. Data-item descriptions must be given for nonstandard labels.

The input-output verbs in the PROCEDURE DIVISION initiate the checking of standard labels on input files and the writing of standard labels on output files. Instructions for writing and checking nonstandard labels must be provided by the programmer. The format of the LABEL RECORDS clause is:

$$\underline{\text{LABEL}} \left\{ \begin{matrix} \underline{\text{RECORDS}} \text{ ARE} \\ \underline{\text{RECORD}} \text{ IS} \end{matrix} \right\} \left\{ \begin{matrix} \underline{\text{STANDARD}} \\ \underline{\text{OMITTED}} \end{matrix} \right\}$$

When the STANDARD option is used, a VALUE clause usually follows to provide identifying information.

The OMITTED option is used for files without labels and for files with nonstandard labels.

The format of the standard label is given in "Appendix B."

VALUE Clause

The VALUE clause specifies the values of label-data-names for standard labels. The $LABEL control card, explained in the publication *IBM 7090/7094 IBSYS Operating System: IBJOB Processor*, Form C28-6389, can be used to provide the same information.

The format of the VALUE clause is:

```
VALUE OF label-data-name-1 IS literal-1
                          [label-data-name-2 IS . . .]
```

An area in storage is reserved for the processing of standard labels. *Label-data-name* is a preassigned name in this area. The value of *label-data-name* is specified by a literal provided by the programmer. Rules for literals are given in "Appendix A." Figures 12 and 13 give the label-data-name, the class, and the form of each literal for input and output files. If the class is alphanumeric, the literal must be enclosed in quotation marks. If the class is numeric, the literal must not be enclosed in quotation marks.

If any of the items is not specified in this clause or on a $LABEL control card for input files, it is not checked when the file is processed. Figure 14 shows information that is automatically written for output files if the value of label-data-name is not specified in the VALUE clause or on a $LABEL control card.

NOTE: In the standard label format, shown in "Appendix B," there is a field named LABEL-IDENTIFIER. A tape is accepted unconditionally as an output tape if the value of LABEL-IDENTIFIER is '1BLANK.' Tapes with the value of LABEL-IDENTIFIER equal to '1HDRbb' are accepted to be written on only if the label field RETENTION-PERIOD shows that the retention period has expired.

Label-Data-Name	Class	Form of Literal
FILE-IDENTIFICATION	alpha-numeric	Eighteen or fewer alphanumeric characters that identify the file.
FILE-SERIAL-NUMBER	alpha-numeric	Five or fewer alphabetic and/or numeric characters (no special characters) with no embedded blanks. This is equivalent to the reel serial number (usually the number on the external casing of the reel of tape).
REEL-SEQUENCE-NUMBER	numeric	Four or fewer digits that specify the number of the reel within a given file. The first reel of the file is reel 1, the second reel is reel 2, etc. The value specified initially is checked against the value in the label of the first reel of the file. Every time a reel is switched, the value specified is increased by one; the new value is then checked against the value in the label of the new reel.

Figure 12. Label Information for Input Files

Label-Data-Name	Class	Form of the Literal
FILE-IDENTIFICATION	alpha-numeric	Eighteen or fewer alphanumeric characters that identify the file.
RETENTION-PERIOD	numeric	Four or fewer digits that specify the number of days that the file is to be saved after its date of creation. The creation date for the file is automatically supplied by the input-output system when the file is created.
FILE-SERIAL-NUMBER	alpha-numeric	Five or fewer alphabetic and/or numeric characters (no special characters) with no embedded blanks. This is equivalent to the reel serial number (usually the number on the external casing of the reel of tape).
REEL-SEQUENCE-NUMBER	numeric	Four or fewer digits that specify the number of the reel within a given file. The value specified is written in the label of the first reel of the file. Every time a reel is switched, the value is increased by one and written in the label of the new reel.

Figure 13. Label Information for Output Files

Label-Data-Name	Information Provided if Item Not Specified
FILE-IDENTIFICATION	The first eighteen characters of the file-name are placed in the label.
RETENTION-PERIOD	The value zero is placed in the label.
FILE-SERIAL-NUMBER	The file-serial-number in the label of the tape being written over is placed in the new label.
REEL-SEQUENCE-NUMBER	The value in the first reel of tape is one. The value is increased by one for each succeeding reel.

Figure 14. Standard Information Provided for Output Files

Description of Logical Records

Following each file description entry are data-item descriptions of the logical records on the file. Each field in the logical record is described. Any of the clauses in the data-item description format may be used in the FILE SECTION; however, the following restrictions have been imposed:

1. The VALUE clause may be used only in *condition-name* entries. A condition-name is a name assigned to a specific value in the range of values a data-item (conditional variable) may assume. Condition-names are identified by the special level-number 88. A condition-name is formed in the same way as a data-name. The rules for name formation are given in "Appendix A." The following example shows the use of the VALUE clause with condition-names.

```
05 MARITAL-STATUS PICTURE IS 9 USAGE IS
   COMPUTATIONAL SYNCHRONIZED RIGHT.
   88 SINGLE VALUE IS 1.
   88 MARRIED VALUE IS 2
   88 DIVORCED VALUE IS 3.
```

The following expressions would be equivalent if used in the PROCEDURE DIVISION.

```
IF SINGLE...
IF MARTIAL-STATUS IS EQUAL TO 1...
```

A conditional variable may not be described as a report or scientific decimal item.

2. The level-number 77 may not be used in the FILE SECTION.

3. The description of logical records on a file must begin with 01 level-numbers. The OCCURS clause must not be used at the 01 level.

4. The REDEFINES clause must not be used at the 01 level for logical records in the same file.

Figure 15 shows data-item descriptions that can be used to describe the logical record PAYROLL.

Working-Storage Section

Areas of storage reserved for processing of data work areas are described in the WORKING-STORAGE SECTION. The only type of entry in this section is the data-item description.

An independent working-storage entry describes a data-item that is not related to other items; that is, it is neither subdivided nor a subdivision of some other item. For example, an independent working-storage item could be used to store an intermediate value of a calculation. These entries are identified by the special level-number 77. The required data-item description clauses for independent working-storage entries are: level-number (77), data-name, PICTURE, or SIZE and CLASS. The OCCURS clause is not meaningful and causes an error at compilation time. The other data-item description clauses are optional and can be used to complete the description of the item if necessary.

Data-items in the WORKING-STORAGE SECTION that are related to each other are grouped into records and described according to the rules for data-item descriptions. The WORKING-STORAGE SECTION can be used to describe entries in tables that are developed in storage or to describe the work area for a record that is read into storage and moved to the area. All clauses used in the data-item description may be used with organized working-storage items. Each record name in the WORKING-STORAGE SECTION (01 level-number) must be unique since it cannot be qualified by a file-name, but subordinate data-names need not be unique if they can be qualified.

In the FILE SECTION, the VALUE clause can be used only in a conditional-name entry; in the WORKING-STORAGE SECTION, the VALUE clause can be used in a condition-name entry and also to specify the initial value of elementary data-items. The general rules

IBM

COBOL PROGRAM SHEET

Form No. X28-1464
Printed in U.S.A.

PAGE 1 3 | PROGRAM TEST-PROGRAM | SYSTEM IBM-7094 | SHEET 2 OF 8
02,0 | PROGRAMMER A.F.A. | DATE 5/28 | IDENT. 73 TEST 01 80

SERIAL 4 6	CONT. 7	A 8 / B 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72
010		01 PAYROLL USAGE IS COMPUTATIONAL.
020		03 MAN-NUMBER PICTURE IS 9999999 SYNCHRONIZED RIGHT.
030		03 HOURS-WORKED.
040		05 REGULAR PICTURE IS 99 SYNCHRONIZED RIGHT.
050		05 OVERTIME PICTURE IS 99 SYNCHRONIZED RIGHT.
060		03 HOURLY-RATE PICTURE IS 99V99 SYNCHRONIZED RIGHT.
070		01 PERSONAL USAGE IS COMPUTATIONAL.
080		02 MARITAL-STATUS PICTURE IS 9 SYNCHRONIZED RIGHT.
090		88 SINGLE VALUE IS 1.
100		88 MARRIED VALUE IS 2.

Figure 15. Sample Data-Item Descriptions in the FILE SECTION

given for the VALUE clause in the data-description apply to its use in the WORKING-STORAGE SECTION. No assumption can be made about the initial value of a working-storage item unless it is specified by a VALUE clause.

The WORKING-STORAGE SECTION begins with the header WORKING-STORAGE SECTION followed by independent items and organized items in that order.

Figure 16 shows coding in the WORKING-STORAGE SECTION.

Constant Section

The CONSTANT SECTION is similar to the WORKING-STORAGE SECTION, except that conditional variables and the OCCURS and REDEFINES clauses may not be used. Each elementary item in the CONSTANT SECTION must have a VALUE clause.

Figure 17 shows sample data-item descriptions in the CONSTANT SECTION.

IBM COBOL PROGRAM SHEET Form No. X28-1464 Printed in U.S.A.

PAGE 3 | PROGRAM TEST PROGRAM | SYSTEM IBM-7094 | SHEET 3 OF 8
030 | PROGRAMMER A.F.A | DATE 5/28 | IDENT. 73 TEST 01 80

SERIAL 4 6 | CONT. 7 | A 8 | B 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72

```
010  WORKING-STORAGE SECTION.
020    77  WEEKLY-PAY USAGE IS COMPUTATIONAL PICTURE IS 99V99
030        SYNCHRONIZED RIGHT.
040    01  EDITING.
050        02 PAY PICTURE IS $$99.99.
060        02 FILLER SIZE IS 5.
070        02 TAX PICTURE IS $9.99.
080    01  WORK-RECORD.
```

Figure 16. Sample Data-Item Descriptions in the WORKING-STORAGE SECTION

```
200  CONSTANT SECTION.
210    77  INTEREST-RATE PICTURE IS V99 VALUE IS .14
220        USAGE IS COMPUTATIONAL SYNCHRONIZED RIGHT.
230    01  TABLE USAGE IS COMPUTATIONAL.
240        02 ONE-DEPENDENT PICTURE IS V99 VALUE IS .14.
250        02 TWO-DEPENDENTS PICTURE IS V99 VALUE IS .10.
```

*A standard card form, IBM electro C61897, is available for punching source statements from this form.

Figure 17. Sample Data-Item Descriptions in the CONSTANT SECTION

Structure of the Procedure Division

The PROCEDURE DIVISION contains instructions for solving a problem; COBOL instructions are written in *statements*, which may be combined to form *sentences*. Groups of sentences form *paragraphs* and paragraphs may be combined to form *sections*.

There are two types of statements; imperative statements consisting of a COBOL verb and its operands, and conditional statements containing conditional expressions, that is, tests for a given condition.

A COBOL sentence is composed of one or more statements and must be terminated by a period. The statement separator THEN may be placed before or after a statement but should not be used at the beginning or end of a sentence.

Several sentences that convey one idea or procedure may be grouped to form a paragraph. A paragraph must begin with a paragraph-name followed by a period. A paragraph is terminated by the next paragraph-name or section-name, or by the end of the PROCEDURE DIVISION. A paragraph may be composed of one or more sentences.

One or more paragraphs form a section. A section must begin with a section-name followed by the word SECTION, followed by a period. Paragraphs need not be grouped into sections; however, section-names are useful for qualifying paragraph-names or for reference. The general term *procedure-name* refers to both paragraph-names and section-names.

Categories of Verbs

COBOL verbs used in imperative statements can be grouped into the following categories.

Declaratives

VERB	FUNCTION
USE	Calls in user routines to supplement automatic input-output error recovery routines and label processing routines.

Input-Output

VERB	FUNCTION
OPEN	Prepares for reading or writing a file.
READ	Causes a logical record to be made available to the program for processing.
WRITE	Causes a logical record to be made available for writing.
CLOSE	Releases buffer area and specifies rewind options.
ACCEPT	Accepts low-volume data from a peripheral device.
DISPLAY	Writes low-volume data on a peripheral device.

Arithmetic

VERB	FUNCTION
COMPUTE	Computes the value of mathematical formulas.
ADD	Adds numbers.
SUBTRACT	Subtracts numbers.
MULTIPLY	Multiplies numbers.
DIVIDE	Divides numbers.

Data Manipulation

VERB	FUNCTION
MOVE	Moves data from one area of storage to another.
EXAMINE	Counts and/or replaces individual characters in a data-item.

Procedure Control

VERB	FUNCTION
GO TO	Transfers control from one part of the program to another.
ALTER	Changes the destination of a GO TO statement.
PERFORM	Allows a series of statements to be executed and control to be returned to the main part of the program.

Compiler Directing

VERB	FUNCTION
EXIT	Supplements, as a dummy verb, the PERFORM statement.
NOTE	Allows comments to be written as part of the program listing.
STOP	Causes a temporary or permanent halt in the program.
ENTER	Provides for communication between programs.

Tests

In the category "TESTS," all procedure statements begin with the word IF. The type of test is given in this chart as the verb type.

VERB	FUNCTION
Relations	Compares the size of two operands.
Sign	Determines whether an operand is positive, negative, or zero.
Class	Determines whether an operand is numeric or alphabetic.
Conditional Variable	Tests if a conditional variable has the value of one of its condition-names.
Switch-Status	Tests the ON or OFF status of an entry key on the computer.

Conditional and Imperative Verbs

The following list shows conditional and imperative statements. Conditional statements contain a condition

that must be tested before the statement is executed. Imperative statements are commands.

IMPERATIVE STATEMENTS	CONDITIONAL STATEMENTS
USE	READ (AT END)
OPEN	ADD
WRITE	SUBTRACT (ON SIZE ERROR)
CLOSE	MULTIPLY (ON SIZE ERROR)
ACCEPT	DIVIDE (ON SIZE ERROR)
DISPLAY	GO TO (DEPENDING ON)
COMPUTE	IF statements
ADD	
SUBTRACT	
MULTIPLY	
DIVIDE	
EXAMINE	
ALTER	
GO TO	
PERFORM	
EXIT	
ENTER	
NOTE	
STOP	

COBOL Verbs

Declaratives

The input-output system automatically handles the checking and writing of standard labels and executes error recovery routines in case of input-output errors.

Procedures in addition to the normal processing by the input-output system may be specified. The declaratives verb USE tells whether the procedure statements that follow it are to be executed in addition to error recovery routines or are to be executed in addition to label processing routines. Any procedure statements (except those containing input-output verbs) may follow a USE statement.

Since these procedure statements are executed at the time an error in reading or writing tape occurs or when labels of files are to be processed, they cannot appear in the regular sequence of procedure statements. Therefore, they must be written at the beginning of the PROCEDURE DIVISION. The word DECLARATIVES identifies the beginning of this section of procedure statements and the declaratives section is ended by the words END DECLARATIVES.

USE Verb

The USE statements identify the procedure statements that follow them. There are two options of the USE statement. Option 1 is used if the statements that follow it give procedures in addition to standard error processing. Option 2 is used if the statements that follow it give procedures in addition to the writing and checking of standard labels. The formats are:

Option 1.

```
section-name SECTION.
  USE AFTER STANDARD ERROR PROCEDURE
    ON { INPUT                          }.
       { file-name-1 [file-name-2 . . .] }
```

Option 2.

```
section-name SECTION.
  USE { BEFORE } STANDARD [{ BEGINNING }]
      { AFTER  }          [{ ENDING    }]
  [{ REEL }] LABEL PROCEDURE ON
  [{ FILE }]
                  { INPUT                          }
                  { OUTPUT                         }
                  { file-name-1 [file-name-2 . . .] }
```

The statements that follow option 1 provide for additional processing after the input-output system executes the standard error recovery routines. When an input-output error occurs, the input-output system executes standard error recovery procedures. If the error cannot be corrected, messages are printed indicating the type of error that has occurred, a core-storage dump is taken, and program execution is terminated. Part II of this publication contains an example that specifies alternate procedures in the declaratives section. If INPUT is specified in option 1, the statements are executed for all input files. Otherwise, the statements are executed for the specified input files; *file-name-1, file-name-2,* etc.

The statements following option 2 provide for processing in addition to standard label processing. The operands in the USE statement determine when and for what files the statements are executed. The following list shows the effect of the operands.

OPTION	STATEMENTS EXECUTED
BEFORE	Before regular label processing.
AFTER	After regular label processing.
BEGINNING	For header labels.
ENDING	For trailer labels.
Option omitted	For both header and trailer labels.
REEL	For labels between first header and last trailer label of the file.
FILE	For first header and/or last trailer label of the file.
Option omitted	For all labels.
INPUT	For all input files.
OUTPUT	For all output files.
file-name-1, file-name-2	For files specified by *file-name-1, file-name-2,* etc.

An example of the use of option 2 is given in Part II of this publication.

NOTE: When option 2 is used, the compiler generates external names (in the form GNXXX) to refer to the USE procedures. The names are placed in the control dictionary, and a job consisting of several programs may have duplicate external names. It may be necessary to

use $NAME cards to rename nonunique external names. The $NAME cards are explained in the publication *IBM 7090/7094 IBSYS Operating System, IBJOB Processor,* Form C28-6389.

The following list shows the order of entries in a declaratives section.

```
PROCEDURE DIVISION.
DECLARATIVES.
section-name-1 SECTION.
    USE . . .
    procedure statements in section-name-1.
section-name-2 SECTION.
    USE . . .
    procedure statements in section-name-2.
section-name-3 SECTION.
        .
        .
END DECLARATIVES.
```

Input-Output Verbs

Input-output verbs are used to transmit data between the computer and peripheral devices. The input-output system automatically handles most input-output functions when an input-output verb is encountered. An OPEN verb initiates reading or writing a file and reserves a buffer area for the file. A READ verb causes one logical record in the buffer area to be made available to the program. The WRITE verb causes one logical record to be made available to be written out of storage from the buffer area. The CLOSE verb closes the buffer area so that it can be used for other files.

The ACCEPT and DISPLAY verbs facilitate the handling of the input and output of data from specified system units, for example, messages to the machine operator that are written on the on-line printer.

OPEN Verb

The OPEN verb initiates the checking or writing of the standard header label preceding the file, causes USE declaratives to be executed, and prepares input-output units to read or write.

A file must be in OPEN status before a READ or WRITE statement can be issued for it. The OPEN verb initializes input-output procedures, but READ and WRITE statements actually obtain and release the records.

The format of the OPEN statement is:

```
OPEN { INPUT file-name-1 [file-name-2 . . .]
     { OUTPUT file-name-1 [file-name-2 . . .]
     { INPUT file-name-1 [file-name-2 . . .]
     {                   OUTPUT file-name-3
     {                          [file-name-4 . . .]
```

File-name-1, file-name-2, etc. are the names the programmer assigned the files in the file description entries in the DATA DIVISION or in the FILE-CONTROL paragraph in the ENVIRONMENT DIVISION.

When a file is opened, it remains in OPEN status until a CLOSE statement for that file is executed. Therefore, a second OPEN statement for a file cannot be executed prior to the execution of a CLOSE statement for that file.

READ Verb

A READ verb causes one logical record of a specified file to be made available to the program. The READ statement includes an instruction for the procedure if the record is the last record on the file. If the file is on more than one reel of tape, the READ verb causes the following operations when the end of the reel is reached.

1. The standard trailer label subroutine and any procedures specified in the declaratives section are executed.
2. A reel switch occurs.
3. The standard header label subroutine and any procedures specified in the declarative section are executed.
4. Checkpoints are written.
5. The next record is made available.

For input files, the standard label subroutines check the values in the labels to ensure that they match the values given in the VALUE clause of the file description entry. If there is an error in the label, a message is printed on-line. The machine operator has the option of ignoring the label error and causing processing to continue. The format of the READ statement is:

```
READ file-name RECORD [INTO data-name]
                    AT END { imperative-statement }
                           { NEXT SENTENCE        }
```

File-name is the name of the file described in the file description entry. Although the name of the file is specified, the READ verb causes the next logical record on the file to be made available. If there is more than one type of record on the file, the programmer must have some means of determining which type has been read.

The INTO *data-name* option causes the record to be put into a working-storage or other record area as well as the input area. It has the same effect as though there were statements to READ the record into the input area and MOVE it to the area specified by *data-name.* The record is available in both the input area and the *data-name* area. *Data-name* must not be the name of a logical record on the file specified by *file-name.* If the format of the *data-name* area is different from that of the input area, moving is performed according to the rules specified for a group MOVE (assumes PICTURE of all X's) without the CORRESPONDING option.

An AT END clause is required in every READ statement. It specifies the procedure to be followed after the last record on the file has been read. After the AT END clause has been executed, an attempt to perform a READ state-

ment without the execution of a CLOSE and a subsequent OPEN for the file constitutes an error except for multireel unlabeled files.

For multireel unlabeled files, the programmer can identify the file in the FILE-CONTROL paragraph in the ENVIRONMENT DIVISION as a multiple reel file and provide a means of communicating to the program, usually by the entry keys or control cards, whether a continuation reel exists for the file. Upon execution of the AT END clause, if it is determined that the last reel of the file has been processed, a CLOSE statement may be given. If it is determined that a continuation reel does exist for the file, the next READ statement causes the input-output system to rewind and unload the reel last read and to switch units if required.

WRITE Verb

A WRITE verb releases a logical record for external storage. If the output file is recorded on more than one reel of tape, the WRITE verb causes the following operations when the end of the reel is reached.

1. The standard trailer label subroutines and procedures specified in the declaratives section are executed.
2. A reel switch occurs.
3. The standard header label subroutine and procedures specified in the declaratives section are executed.
4. Checkpoints are written.

For output files, standard label subroutines cause the values specified in the VALUE clause of the file description entry to be written in the label.

The format of the WRITE verb is:

```
WRITE record-name [FROM data-name]
```

Record-name is the name assigned to the logical record in the FILE SECTION (a data-item description with a 01 level-number).

The FROM *data-name* option specifies that the record is to be moved from a working-storage area or other record area named *data-name* to the output area *record-name* and then released for external storage. If the format of *data-name* differs from *record-name*, moving is performed according to the rules for a group MOVE (assumed PICTURE of all X's) without the CORRESPONDING option.

After the WRITE statement is executed, the information in *record-name* cannot be referred to; but if the FROM option is used, the information in *data-name* is still available.

CLOSE Verb

The CLOSE verb initiates the closing of the file and releases the buffer area for further use by the program. The format of the CLOSE statement is:

```
Option 1.
CLOSE file-name-1 [ WITH { NO REWIND } ]
                         { LOCK      }
                                        [file-name-2 . . .]

Option 2.
CLOSE file-name-1 REEL [file-name-2 . . .]
```

Option 1 is used to terminate processing on a file. When a CLOSE file-name statement is given, the specified options are executed for the current reel of the file and no further processing on the file is permitted until an OPEN statement is given. The following list shows the effect of the options in the CLOSE file-name statement:

OPTION	ACTION FOR THE CURRENT REEL OF THE FILE
LOCK	rewinding and unloading
NO REWIND	no rewinding
neither option specified	rewinding

Option 2, the REEL option, is used when the file is on more than one reel of tape, and the programmer wants to stop reading or writing the records on one reel and go on to the next reel. LOCK, that is, rewinding and unloading, is assumed when this option is used. On labeled input files, the trailer label is not checked; but the header label on the next reel is checked and statements in the declaratives section are executed. If this option is used for the last reel of an input file, an error is probable for unlabeled files and certain for labeled files. On labeled output files, both trailer and header labels are written and the statements in the declaratives section are executed. Checkpoints specified (in the I-O-CONTROL paragraph of the ENVIRONMENT DIVISION) for either input or output files are not taken if the REEL option is used.

If a file is assigned to a system unit in the ASSIGN TO clause in the ENVIRONMENT DIVISION, the CLOSE options for the file may be overridden by the CLOSE options of the system unit.

Figures 18 and 19 show the operations initiated by the input-output verbs, OPEN, READ, WRITE, and CLOSE.

ACCEPT Verb

The ACCEPT verb permits low-volume data to be read from the card reader or the system input unit (SYSIN1). The format of the ACCEPT statement is:

```
ACCEPT data-name [FROM SYSIN1]
```

The USAGE of *data-name* must be DISPLAY and it may not be a report, scientific decimal, or floating-point item. *Data-name* may not exceed 72 characters. The

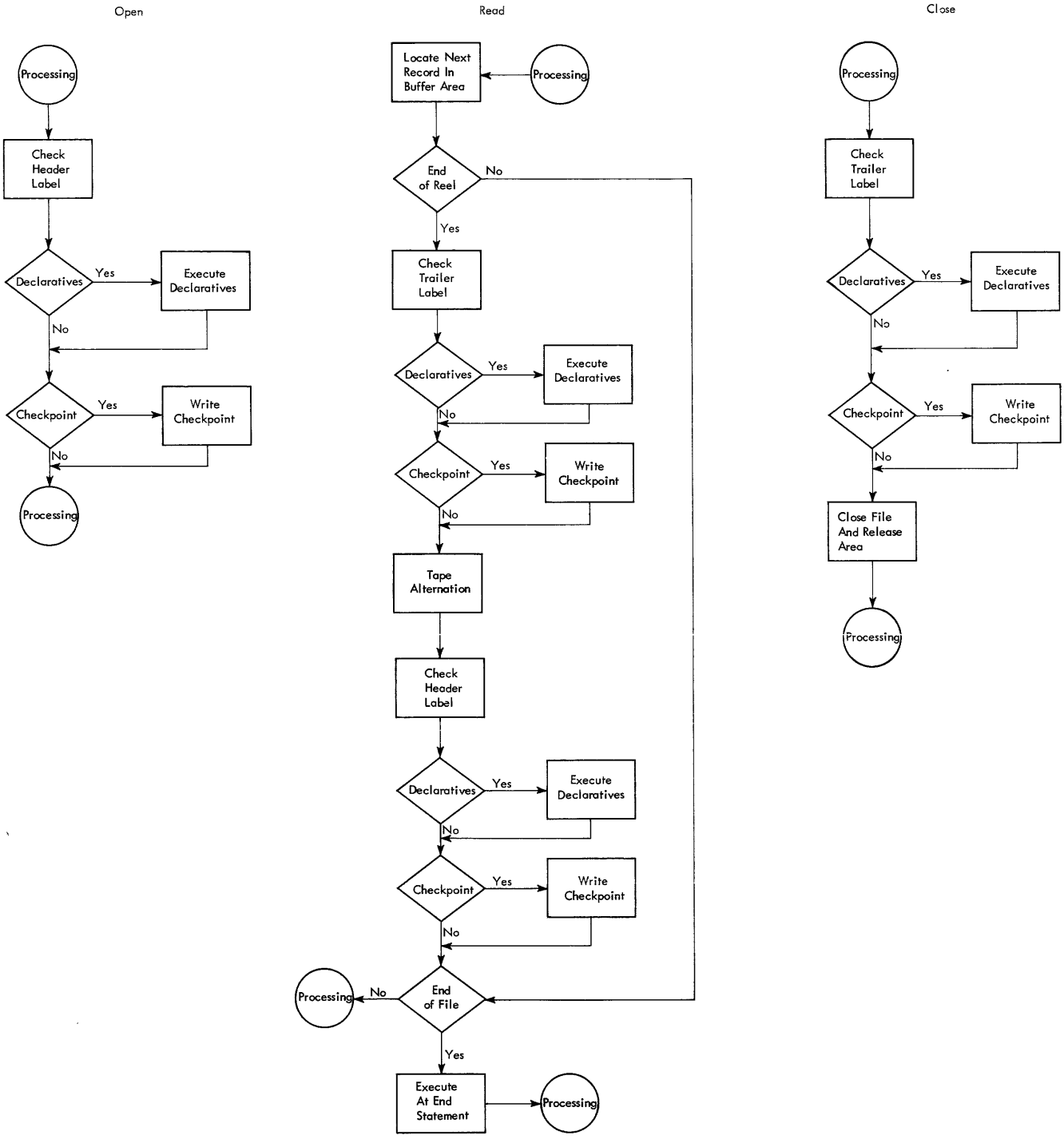


Figure 18. Operations Initiated by Input-Output Verbs on Input Files

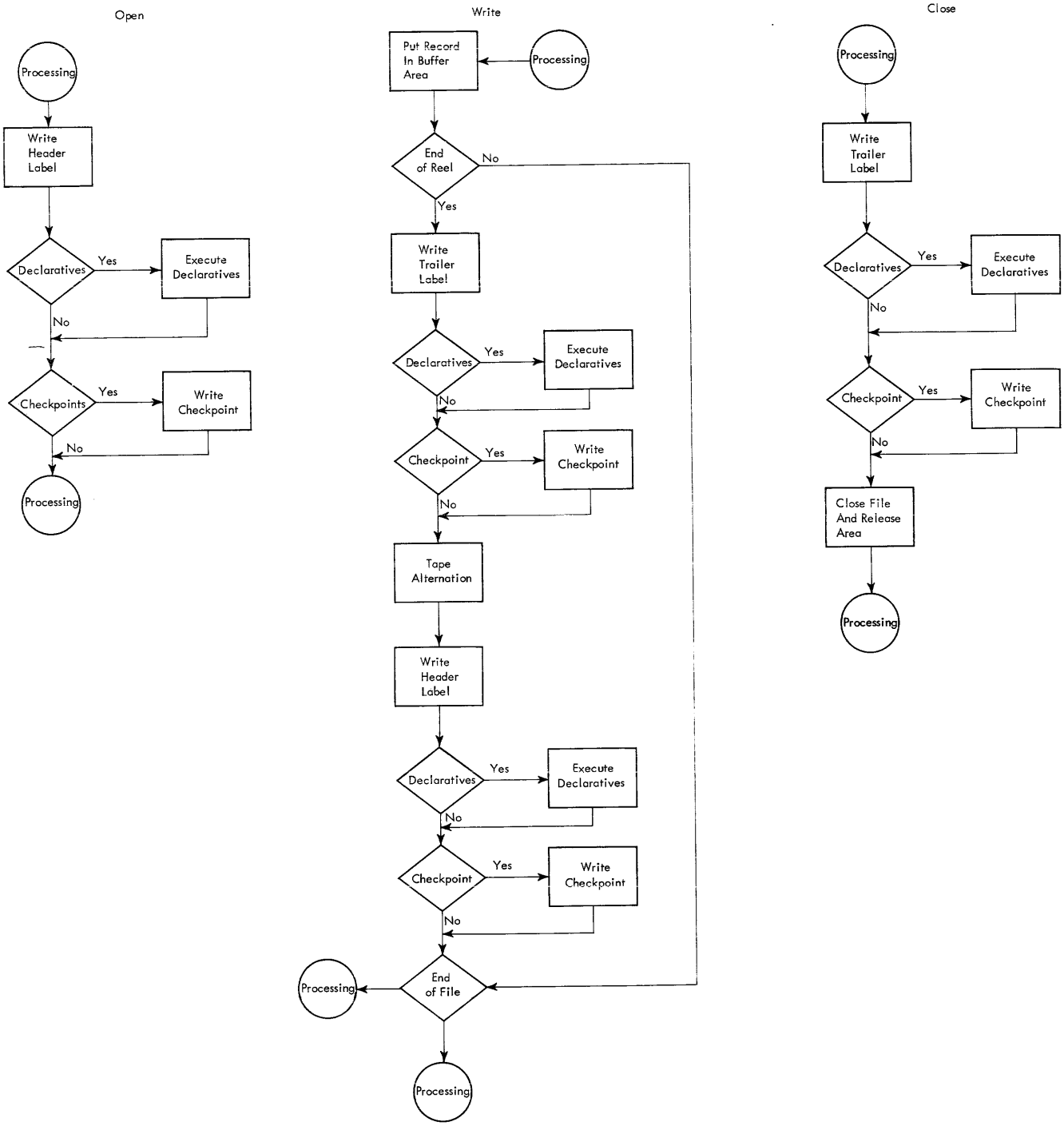


Figure 19. Operations Initiated by Input-Output Verbs on Output Files

length of *data-name* may not vary in the program, that is, the data-item description entry may not contain an OCCURS DEPENDING ON clause.

DISPLAY Verb

The DISPLAY verb causes low-volume data to be written on either the on-line printer or the system output unit (SYSOU1). The format of the DISPLAY statement is:

```
DISPLAY { literal-1           } [ { literal-2           } ... ]
        { data-name-1         }   { data-name-2         }
        { figurative-constant-1 }   { figurative-constant-2 }
                                                [UPON SYSOU1]
```

The standard display device is the on-line printer. If UPON SYSOU1 is specified, the display device is the system output unit. Normally, a line of displayed information is 72 characters long. If UPON SYSOU1 is specified and SYSOU1 is a magnetic tape unit, the information displayed is in strings of 120 characters, that is, the "print line" is 120 characters long. Automatic carriage control for off-line printing of output on SYSOU1 is provided.

When DISPLAY is followed by multiple operands, the data constituting the first operand is made the first set of characters on the first line, the data constituting the second operand is made the second set of characters on the first line, and so forth, until the print line is filled. Then the line is displayed. The only operands that can be split so that parts of the item appear on two or more successive print lines are either group items or elementary items whose PICTURE clause is either A(n) alphabetic or X(n) alphanumeric. If there is not enough room in the print line for any other type of item, the current print line is filled with spaces and displayed. The pending item is the first set of characters on the new print line.

Negative NUMERIC COMPUTATIONAL data-items and all NUMERIC DISPLAY data-items are prepared for external output with the sign indicated by an overpunch in the rightmost character position. The printer recognizes these as alphanumeric characters and prints them accordingly. A complete list of the alphanumeric characters corresponding to a digit with a sign overpunch is given in "Appendix B."

If a file is assigned to SYSOU1, data written on the file (WRITE statement) and items displayed on SYSOU1 do not necessarily appear on the listing in the order written. Output from a DISPLAY statement is produced immediately, but output from a WRITE statement is put in a buffer area and written by the input-output system when the buffer is full. If SYSOU1 is used both for a DISPLAY device and for an output file, the automatic carriage control feature is subject to interference.

The special register TALLY is a valid operand in a DISPLAY statement.

Arithmetic Verbs

Arithmetic verbs are used for computations. Each operation can be individually specified with the verbs ADD, SUBTRACT, MULTIPLY, and DIVIDE; or the operations can be combined in a formula expressed in terms of mathematical symbols. The COMPUTE verb is used with formulas.

The FORTRAN IV mathematical library subroutines can be used in a COBOL program to perform other algebraic operations. A description and example are given in Part II of this publication.

COMPUTE Verb

The COMPUTE verb allows arithmetic operations to be specified in a formula. It may be used instead of the ADD, SUBTRACT, MULTIPLY, and DIVIDE statements, which are internally changed to COMPUTE statements by the compiler.

The format of the COMPUTE statement is:

```
COMPUTE data-name-1 [ROUNDED] = { formula      }
                                { data-name-2 }
                                { literal     }
                    [ON SIZE ERROR imperative-statement]
```

Data-name-1 must be the name of an elementary data-item described in DATA DIVISION or the special register TALLY; it cannot be a literal. The result of the calculation is stored in *data-name-1*.

If *data-name-2* or *literal* is specified, *data-name-1* is set equal to *data-name-2* or *literal*. In this case, *data-name-1* can be any legitimate receiving field for a MOVE from *data-name-2* or *literal*. Legitimate receiving fields are shown in Figure 22.

Formula can be any combination of numeric literals, data-names, and arithmetic symbols. There are five arithmetic symbols used in formulas in COBOL. The symbols and their meanings are:

SYMBOL	MEANING
+	Add
−	Subtract
*	Multiply
/	Divide
**	Exponentiate

The order in which the operations are to be performed may be indicated by parentheses. If parentheses are nested, expressions in the innermost parentheses are evaluated first. The following examples show the order of operations indicated by parentheses:

Example 1:

```
COMPUTE RESULTS = 3 * (2 + 5) * (8 / 2)
```

results in

```
RESULT = 3 * 7 * 4 = 84
```

Example 2:

```
COMPUTE RESULT = 3 + (4 * (7 + 8)) + 2
```

results in

```
RESULT = 3 + (4 * 15) + 2 = 3 + 60 + 2 = 65
```

If there are no parentheses, the expression is evaluated in the following order: first, exponentiation, then multiplication and division, and last, addition and subtraction. Expressions on the same level are evaluated from left to right. For example, A / B * C is the same as (A / B) * C. The following example shows the evaluation of an expression in which the order is not indicated by parentheses.

Example:

```
COMPUTE RESULT = 2 + 10 / 5 + 2 ** 3 * 4 - 6
```

results in

```
RESULT = 2 + (10 / 5) + ((2 ** 3) * 4) - 6
       = 2 + 2 + (8 * 4) - 6
       = 2 + 2 + 32 - 6
       = 30
```

Exponentiation of either a negative variable or literal is allowed only if the exponent is either a literal or data-name having an integral value. If the exponent is other than a non-negative integer, the result of the exponentiation is zero.

It should be noted that the operational symbols must be preceded and followed by a space.

Decimal point alignment is automatically supplied by the compiler. If the number of decimal places (those to the right of the decimal point) in the result is greater than the number of decimal places in the description of *data-name-1*, excess decimal places are truncated unless the ROUNDED option is specified. When ROUNDED is specified, the least significant digit of the result in *data-name-1* is increased by 1 if the most significant digit of the excess is greater than or equal to 5.

Whenever the number of integral places (those to the left of the decimal point) in the result exceeds the number of integral places in *data-name-1*, a size error condition arises. When ON SIZE ERROR is not specified and a size error occurs, the value of *data-name-1* is unpredictable. When ON SIZE ERROR is specified and a size error occurs, the value of *data-name-1* is unpredictable, but the statement following the words ON SIZE ERROR is executed.

ADD Verb

The ADD verb is used to add two or more numeric data-items. The format of the ADD statement is:

Option 1.

```
ADD {literal-1   }[{literal-2   }...][{TO    }
    {data-name-1}[{data-name-2}...][{GIVING} data-name-n]
    [ROUNDED] [ON SIZE ERROR imperative-statement]
```

Option 2.

```
ADD CORRESPONDING data-name-1 TO data-name-2
    [ROUNDED] [ON SIZE ERROR imperative-statement]
```

Option 1 is used to add two or more elementary items. *Data-name-1*, *data-name-2*, etc., may be numeric elementary data-items described in the DATA DIVISION (e.g., with the numeric form of the PICTURE clause) or may be the special register TALLY. The literals must be numeric.

If the GIVING option is used, the sum of the items preceding *data-name-n* is stored in *data-name-n*. Since *data-name-n* is not an operand, it may contain editing symbols in the PICTURE clause of its data-item description.

If the TO option is used, the sum of all the items, including *data-name-n*, is stored in *data-name-n*. In this case, *data-name-n* is used in the calculation and must not contain editing symbols. If neither TO nor GIVING is specified, TO is assumed.

Option 2, ADD CORRESPONDING, is used to add the elementary items in a group to matching elementary items in another group. *Data-name-1* and *data-name-2* are the names of the groups containing the elementary items to be added. Two elementary items match if their names are the same, including all qualification up to (not including) *data-name-1* and *data-name-2*. An item in the group *data-name-1* is added to an item in the group *data-name-2* that has the same name; the result is stored in the group *data-name-2*.

In both option 1 and option 2, ROUNDED and SIZE ERROR have the same meanings as in the COMPUTE statement.

Figure 20 shows the effect of an ADD CORRESPONDING statement.

ADD CORRESPONDING WEEKLY-RECORDS TO YEARLY-RECORDS

Data-names in group data-name-1 (WEEKLY-RECORDS)	Values Before ADD CORRESPONDING	Values After ADD CORRESPONDING	Data-names in group data-name-2 (YEARLY-RECORDS)	Values Before ADD CORRESPONDING	Values After ADD CORRESPONDING
03 WEEKLY-RECORDS	—	—	02 YEARLY-RECORDS	—	—
04 HOURS-WORKED	40	40	04 HOURS-WORKED	120	160
04 PAYROLL	—	—	04 PAYROLL	—	—
05 PAY	100	100	06 PAY	300	400
05 NET-PAY	97	97	04 TAX-RATE	03	03
04 TAX	3	3	04 TAX	9	12

Figure 20. Example of ADD CORRESPONDING

Subtract Verb

The SUBTRACT verb is used to subtract one, or a sum of two or more, numeric data-items from a specified item. The format of the SUBTRACT statement is:

```
Option 1.
SUBTRACT { literal-1   } [ { literal-2   } ... ]
         { data-name-1 }   { data-name-2 }
  FROM { literal-n   } [GIVING data-name-m]
       { data-name-n }
    [ROUNDED] [ON SIZE ERROR imperative-statement]

Option 2.
SUBTRACT CORRESPONDING data-name-1 FROM
                                   data-name-2
    [ROUNDED] [ON SIZE ERROR imperative-statement]
```

Option 1 is used to subtract one or more elementary items from another elementary item. When dealing with multiple subtrahends, the effect of the subtraction is as though the subtrahends were first added together, and the sum then subtracted from the minuend.

Data-name-1 through *data-name-n* must be elementary numeric data-items described in the DATA DIVISION, or the special register TALLY. *Literal-I* through *literal-n* must be numeric literals.

If the GIVING option is used, the result is stored in *data-name-n*, which may contain editing symbols since it is not used in the calculation. If the GIVING option is not specified, the result of the subtraction is stored in *data-name-n* which, therefore, may not be a literal. *Data-name-n* may not contain editing symbols since it is used as an operand.

The effect of the ROUNDED and ON SIZE ERROR options is the same as for the COMPUTE statement.

Option 2, the CORRESPONDING option, is the same as the ADD CORRESPONDING option, except that matching items are subtracted rather than added.

MULTIPLY Verb

The MULTIPLY verb is used to multiply two numeric data-items. The format of the MULTIPLY statement is:

```
MULTIPLY { data-name-1 } BY { data-name-2 }
         { literal-1   }    { literal-2   }
  [GIVING data-name-3] [ROUNDED]
                  [ON SIZE ERROR imperative-statement]
```

If the GIVING option is specified, the result of the multiplication is stored in *data-name-3*, which may contain editing symbols. If the GIVING option is not specified, the result of the multiplication is stored in *data-name-2*, which, therefore, must neither be a literal nor contain editing symbols.

Data-name-1 and *data-name-2* must be either numeric elementary items containing no editing symbols or they may be the special register TALLY. *Literal-1* and *literal-2* must be numeric literals.

The ROUNDED and ON SIZE ERROR options are the same as for the COMPUTE statement.

The statement MULTIPLY RATE BY TIME GIVING DISTANCE has the same effect as COMPUTE DISTANCE = RATE * TIME.

DIVIDE Verb

The DIVIDE verb is used to divide one numeric data-item by another. The format of the DIVIDE statement is:

```
DIVIDE { data-name-1 } INTO { data-name-2 }
       { literal-1   }      { literal-2   }
  [GIVING data-name-3] [ROUNDED]
                  [ON SIZE ERROR imperative-statement]
```

When the GIVING option is specified, the result of the division is stored in *data-name-3*, which may contain editing symbols. When the GIVING option is not specified, the result is stored in *data-name-2* (not a literal), which must not contain editing symbols.

Data-name-1 and *data-name-2* must be either numeric elementary items containing no editing symbols, or the special register TALLY. *Literal-1 and literal-2* must be numeric literals.

The ROUNDED and ON SIZE ERROR options follow the rules for the COMPUTE statement.

Since division by zero constitutes a size error, it is wise to test for a zero condition before dividing.

The statement DIVIDE TOTAL-COST INTO QUANTITY GIVING ITEM-PRICE is equivalent to COMPUTE ITEM-PRICE = QUANTITY / TOTAL-COST.

Data Manipulation Verbs

The MOVE verb moves data from one area of core storage to another. The EXAMINE verb is used to count the occurrences of, and/or to replace, a specified character in a data-item.

MOVE Verb

The MOVE verb is used to move data from one area of core storage to one or more other areas. Both the source area and the receiving area(s) must be described in the DATA DIVISION. The data is stored in conformity with the description of the receiving area; therefore, this verb can be used to convert data from one form to another. For example, a numeric item can be edited by moving it to a data-item described by the report form of the PICTURE clause.

The format of the MOVE statement is:

```
Option 1.
MOVE { data-name-1 } TO data name 2 [data name 3 . . .]
     { literal     }
Option 2.
MOVE CORRESPONDING data-name-1 TO data-name-2
                                  [data-name-3 . . .]
```

Option 1 is the simple MOVE statement. The value of the data in the source field, *data-name-1* or *literal*, is unchanged by the MOVE statement. It is stored in accordance with the description of the receiving fields, *data-name-2*, *data-name-3*, etc.

When numeric items (COMPUTATIONAL or DISPLAY) are moved to numeric fields, they are aligned by decimal points with zero-filling or truncation on either end, as required. A warning message is given by the compiler if there is a possibility that significant digits will be lost through truncation when the program is executed.

When numeric COMPUTATIONAL items are moved to numeric DISPLAY fields, they are converted from binary to BCD; that is, USAGE becomes DISPLAY. Since COMPUTATIONAL items are assumed to be signed, the DISPLAY PICTURE is preceded by an S.

When numeric items (COMPUTATIONAL or DISPLAY) are moved to report fields for editing, they are aligned as specified in the report field. (Editing is explained under the report form of the PICTURE clause.) After the item is edited, it is treated as alphanumeric data when referred to in the program.

When numeric items are moved to alphanumeric fields, they are stored in the field from left to right and the signs are dropped. Nonsignificant digits are truncated if the item is too long. Trailing blanks fill in the field if the item is too short.

Nonnumeric data is placed in the receiving field from left to right. If the source field is shorter than the receiving field, the extra positions in the receiving field are filled with spaces. If the source field is longer than the receiving field, the operation is terminated when the receiving field is filled, and a warning message is given. Nonnumeric data must not be moved to numeric fields.

Figure 21 shows the effect of MOVE statements.

A group item may be moved to an elementary item; an elementary item may be moved to a group item. If either is done, the data is treated as nonnumeric data and placed in the receiving field from left to right.

Figure 22 shows the kind of data that can be moved from one area of storage to another.

Option 2, the CORRESPONDING option, is used to move group items. Items in the group *data-name-1* are moved to items in the group *data-name-2* (*data-name-3*, etc.) that have the same name. The CORRESPONDING option has the same effect as moving the elementary items individually.

As in the simple MOVE, the CORRESPONDING option causes the data to be stored in conformity with the receiving field. An item in the group *data-name-1* is moved to a field in *data-name-2* if the names are the same including all qualification up to (but not including) *data-name-1* and *data-name-2*. At least one of the items in a matching pair must be elementary. This option may not be used with data-items having level-numbers of 77 or 88. Neither *data-name-1* nor *data-name-2* may contain an OCCURS clause. If a REDEFINES clause has been used, the data description for the name referred to by the MOVE CORRESPONDING statement is used.

Figure 23 shows the effect of a MOVE CORRESPONDING statement. The noncorresponding items in the source area are not moved and the noncorresponding items in the receiving area are not affected.

Source Area			Receiving Area		
PICTURE	Data Before MOVE	Data After MOVE	PICTURE	Data Before MOVE	Data After MOVE
X(4)	1.23	1.23	X(6)	ABCDEF	1.23bb
9V99	1.23	1.23	X(6)	13.231	123bbb
999	123	123	X(6)	bbbbbb	123bbb
S9(6)	−000199	−000199	X(4)	bOUT	0001
9(4)V9(5)	1234.56789	1234.56789	X(72)	bbb. . . b	123456789bb. . . b
99V99	12.34	12.34	99V99	98.76	12.34
99V99	12.34	12.34	99V9	98.7	12.3
9V9	1.2	1.2	99V999	98.654	01.200
X(3)	A2B	A2B	XXXXXX	Y9X8W	A2Bbbb
99V99	12.34	12.34	$ZZZ9.99	$8765.43	$ 12.34

Figure 21. Effect of MOVE Statements

Source Field Type	Receiving Field Type: Group Items (1)	Elementary Items: Alphabetic (2)	Alphanumeric	Numeric COMPUTATIONAL	Numeric DISPLAY	Report	Floating-point COMPUTATIONAL	Scientific Decimal
Group Items (1)	yes							
Elementary Items		yes	yes	no	no	no	no	no
Alphabetic	yes	yes	yes	no	no	no	no	no
Alphanumeric	yes	yes	yes	no	no	no	no	no
Numeric COMPUTATIONAL	yes	no	yes	yes	yes	yes	yes	yes
Numeric DISPLAY	yes	no	yes	yes	yes	yes	yes	yes
Report	yes	no	yes	no	no	no	no	no
Floating-point COMPUTATIONAL	yes	no	yes	yes	yes	yes	yes	yes
Scientific Decimal	yes	no	yes	yes	yes	yes	yes	yes
Figurative Constants								
ZERO[S] or ZEROES	yes	no	yes	yes	yes	yes (3)	yes	yes (3)
SPACE[S]	yes	yes	yes	no	no	no (4)	no	no
LOW-VALUE[S]	yes	no	yes	no	no	no	no	no
HIGH-VALUE[S]	yes	no	yes	no	no	no	no	no
QUOTE[S]	yes	no	yes	no	no	no	no	no
ALL...	yes	yes (2)	yes	no	yes (5)	no	no	no

(1) Group items are treated as having a PICTURE of all X's.
(2) All characters must be alphabetic.
(3) Zero suppression takes place if specified.
(4) A warning message is given.
(5) The character must be numeric.

Figure 22. Permissible Source and Receiving Fields in MOVE Statements

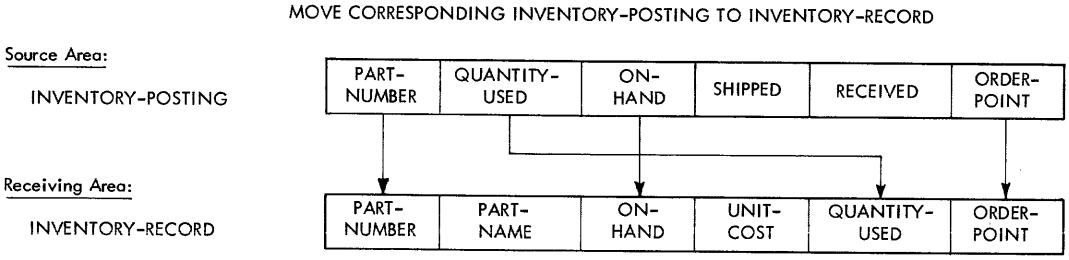


Figure 23. Effect of MOVE CORRESPONDING Statement

EXAMINE Verb

The EXAMINE verb is used either to count the number of times a specified character appears in a data-item and/or to replace a character with another character. The format of the EXAMINE statement is:

```
EXAMINE date-name
{ TALLYING { ALL     } literal-1
           { LEADING }
           { UNTIL FIRST }
             [REPLACING BY literal-2] }
{ REPLACING { ALL     }
            { LEADING }
            { UNTIL FIRST } literal-1 BY literal-2
            { FIRST   }                          }
```

The EXAMINE verb causes *data-name* to be scanned from left to right. When the TALLYING option is used, a count is made of the occurrences of *literal-1* in *data-name*. The count replaces the value of the special register TALLY. The word TALLY is the preassigned name of a special register that can hold five decimal digits.

The REPLACING option (used alone or with the TALLYING option) causes *literal-1* to be replaced by *literal-2* in *data-name*. If the REPLACING option is used with the TALLYING option, the characters counted are the ones that are replaced, that is, the option chosen for TALLYING applies to REPLACING.

The word ALL specifies that each occurrence of *literal-1* in the *data-name* is counted and/or replaced by *literal-2*.

The word LEADING specifies that *literal-1* is counted and/or replaced by *literal-2* only until a character different from *literal-1* is encountered.

The word UNTIL FIRST specifies that characters other than *literal-1* are counted and/or replaced by *literal-2* until the first occurrence of *literal-1*.

The word FIRST (used only with the REPLACING option) specifies that only the first occurrence of *literal-1* is to be replaced by *literal-2*.

The EXAMINE statement can be used only with DISPLAY data-items. Any literal used must be a member of the character set associated with the CLASS of *data-name*; e.g., if the CLASS of *data-name* is NUMERIC, the literal must be numeric and not enclosed in quotation marks. The literals must be single characters. A figurative constant may be specified instead of the literals and it will be considered to be a single-character literal. Literals and figurative constants are explained in "Appendix A."

Procedure Control Verbs

Normally, instructions in the PROCEDURE DIVISION are executed sequentially. The procedure control verbs allow alterations in the sequence. The GO TO verb causes a transfer to another part of the program. The ALTER verb is used to change the destination of a GO TO statement. The PERFORM verb causes a series of statements to be executed a specified number of times.

GO TO Verb

The GO TO verb allows a transfer from one part of the program to another. The format of the GO TO statement is:

```
Option 1.
paragraph-name. GO TO

Option 2.
GO TO procedure-name-1

Option 3.
GO TO procedure-name-1 procedure-name-2
     [procedure-name-3 . . .] DEPENDING ON data-name
```

In option 1, the destination of the GO TO verb is not given. It must be specified by an ALTER statement before the GO TO statement is executed. *Paragraph-name* is required and the paragraph may contain only the words GO TO.

When option 2 is used, the next statement to be executed after the GO TO statement is the first statement in *procedure-name-1*. The ALTER verb can be used to change the destination of the GO TO statement. If this is done, the GO TO statement must appear in a paragraph by itself.

Option 3 provides another method of transferring to different parts of the program. The transfer is to the 1st, 2nd, . . . , nth *procedure-name* as the value of *data-name* is 1, 2, . . . , n. In order to transfer, *data-name* must have a positive integral value; if the value is anything other than the integers 1, 2, . . . , n, then no transfer is executed and control passes to the next statement in the normal sequence.

ALTER Verb

The ALTER verb changes the transfer point specified in a GO TO statement. The format of the ALTER statement is:

```
ALTER paragraph-name-1 TO PROCEED TO
     paragraph-name-2 [paragraph-name-3 TO PROCEED TO
                                paragraph-name-4 . . .]
```

Paragraph-name-1, paragraph-name-3, etc., are the names of paragraphs consisting of only a nonconditional GO TO statement, that is, option 1 or 2 of the GO TO verb.

Paragraph-name-2, paragraph-name-4, etc., are the names of paragraphs to which the GO TO statement causes a transfer after the ALTER statement is executed.

For example, in the coding given below, a transfer is made to the paragraph named SORT the first time through the program and to the paragraph named MERGE after the statement at REVERSE is executed.

```
P1. ALTER P3 TO PROCEED TO SORT.
     .
     .
P3. GO TO
     .
     .
REVERSE. ALTER P3 TO PROCEED TO MERGE.
```

PERFORM Verb

The PERFORM verb is used to depart from the normal sequence of procedures in order to execute one statement, or a series of statements, a specified number of times. After the statements are executed, control is returned to the statement after the PERFORM statement. The format of the PERFORM statement is:

```
Option 1.
PERFORM procedure-name-1 [THRU procedure-name-2]

Option 2.
PERFORM procedure-name-1 [THRU procedure-name-2]
                          { data-name } TIME[S]
                          { integer   }
```

Option 3.

```
PERFORM procedure-name-1 [THRU procedure-name-2]
                                    UNTIL condition
```

Option 4.

```
PERFORM procedure-name-1 [THRU procedure-name-2]
  VARYING data-name-1 FROM { data-name-2 }
                           { literal-1   }
                    BY { data-name-2 } UNTIL condition
                       { literal-2   }
```

Option 5.

```
PERFORM procedure-name-1 [THRU procedure-name-2]
  VARYING subscript-name-1 FROM { data-name-1 }
                                { integer-1   }
                    BY { data-name-2 } UNTIL condition-1
                       { integer-2   }
          [ AFTER subscript-name-2
  FROM { data-name-3 } BY { data-name-4 }
       { integer-3   }    { integer-4   }
                                    UNTIL condition-2
               [AFTER subscript-name-3
  FROM { data-name-5 } BY { data-name-6 }              ]
       { integer-5   }    { integer-6   }
                                    UNTIL condition-3]
```

All of the options begin with PERFORM *procedure-name-1* [THRU *procedure-name-2*]. *Procedure-name-1* is the location of the statement to be executed after the PERFORM statement. If *procedure-name-2* is specified, the series of statements to be executed include the statements at *procedure-name-1* through the statements at *procedure-name-2*.

If *procedure-name-2* is not specified, statements following *procedure-name-1* are executed until another procedure-name of the same level as *procedure-name-1* is encountered; that is, if *procedure-name-1* is a paragraph-name, all the statements in the paragraph are executed; if *procedure-name-1* is a section-name, all the statements in the section are executed.

The last statement in the sequence to be performed must not be a GO TO statement. The EXIT statement may be used as the last statement of the sequence if there are several places in the sequence that transfer control to the last statement (and, hence, to the statement following the PERFORM statement).

Option 1 is a simple PERFORM statement. The specified sequence of statements is executed, and control is returned to the statement immediately following the PERFORM statement.

Option 2 is the TIMES option. The number of times the series of statements is executed is specified by an integer, or by a data-name that contains a nonnegative integer.

Option 3 is the UNTIL option. The series of statements is executed until the specified condition is met.

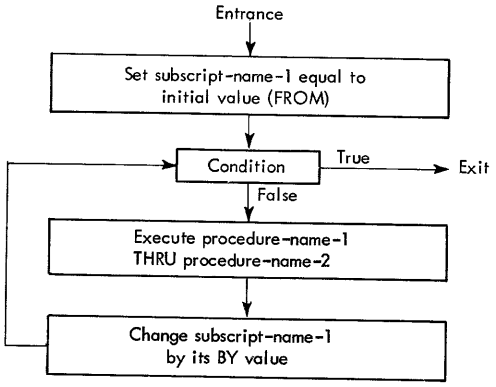
Option 4 is the VARYING option. *Data-name-1* is increased or decreased by the value of *data-name-3* (or *literal-2*) each time the series of procedures is executed until the specified condition is met. *Data-name-2* (or *literal-1*) gives the initial value of *data-name-1*. *Data-name-2*, *data-name-3*, *integer-1*, and *integer-2* must be numeric, but they need not be integers.

Option 5 is the VARYING subscript-name option. The subscripts are incremented in a nested fashion each time the series of statements is executed until the conditions are met. A maximum of three subscripts can be varied.

When only one subscript is being varied, the operation is the same as that of the VARYING data-name option (option 4). Figure 24 shows how the PERFORM statement with one subscript is evaluated.

When two subscripts are varied, the value of *subscript-name-2* goes through a complete cycle (FROM, BY, UNTIL) each time that *subscript-name-1* is increased or decreased by its BY value. The PERFORM is completed as soon as *condition-1* is true. Figure 25 shows how a PERFORM statement with two subscripts is evaluated.

When three subscripts are used, the value of *subscript-name-3* goes through a complete cycle each time that *subscript-name-2* is changed by its BY value. Furthermore, *subscript-name-2* goes through a complete cycle each time that *subscript-name-1* is changed by its BY value. The PERFORM is completed when *condition-1* is true. The flow chart in Figure 26 shows how option 5 of the PERFORM verb with three subscripts is evaluated.

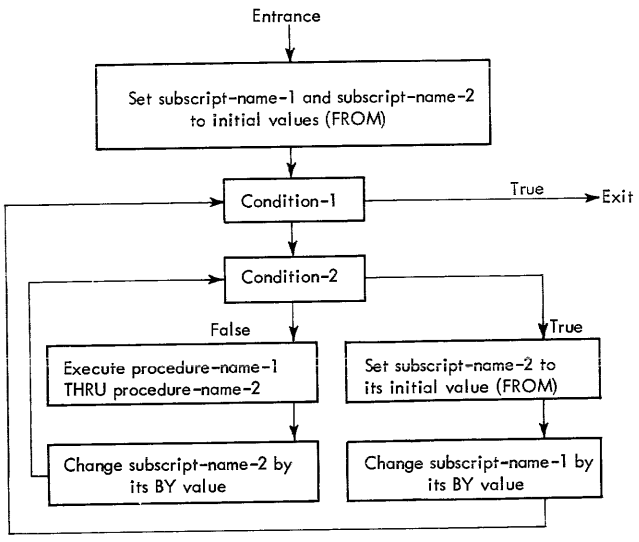


NOTE: The value of subscript-name-1 is increased or decreased before the condition is tested. For example, the statement

```
PERFORM procedure-name-1 THRU procedure-name-2
  VARYING subscript-name-1 FROM 1 by 1 UNTIL
  subscript-name-1 = 6
```

will cause the statements from procedure-name-1 through procedure-name-2 to be executed five times.

Figure 24. PERFORM Statement with One Subscript

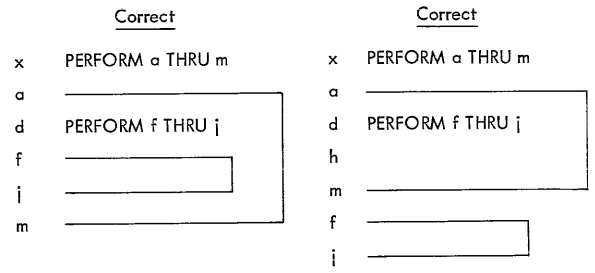


NOTE: The subscripts are increased or decreased before the condition is tested. When the statements under control of the PERFORM verb are executed the specified number of times, subscript-name-2 has its initial value (FROM).

Figure 25. PERFORM Statement with Two Subscripts

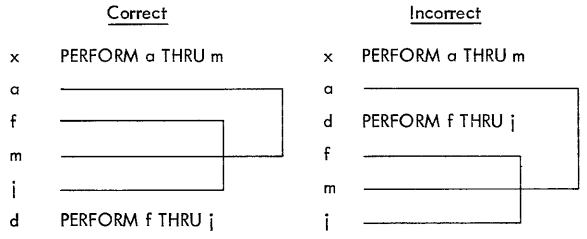
If a series of statements referred to by a PERFORM statement includes another PERFORM statement, the series associated with the included PERFORM statement must be either totally included in or totally excluded from the series associated with the first PERFORM statement.

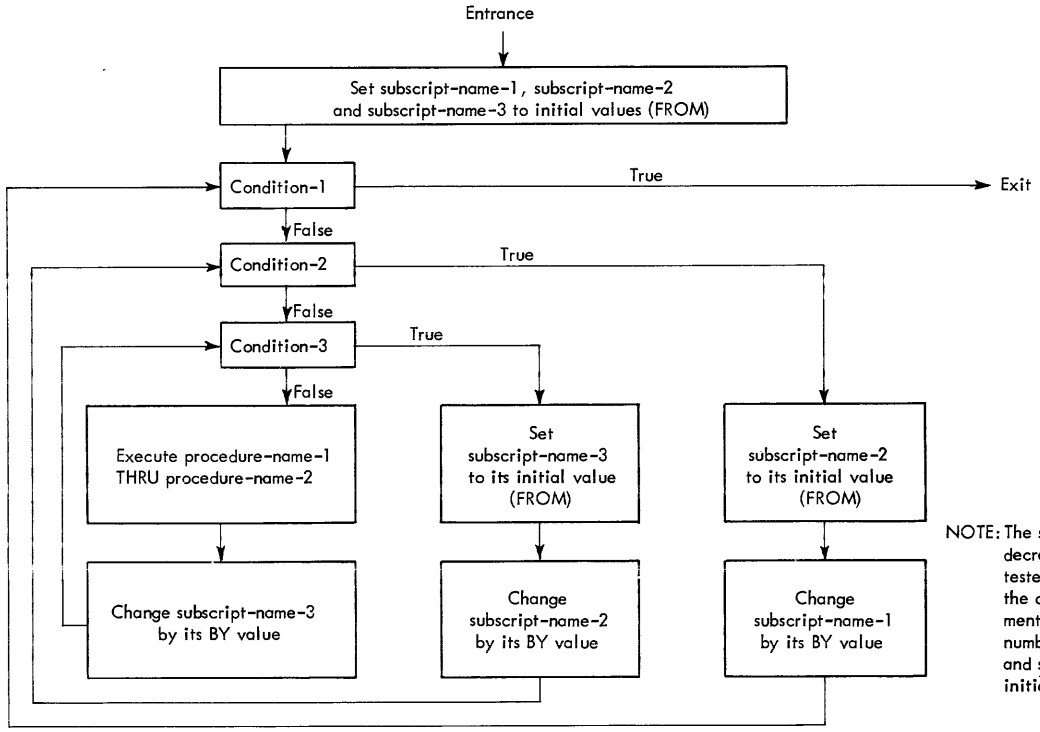
For example, the following diagrams show permissible conditions:



The series of procedures associated with a PERFORM statement may overlap or intersect the series associated with another PERFORM statement, provided neither sequence includes the PERFORM statement associated with the other sequence.

For example:





NOTE: The subscripts are increased or decreased before the condition is tested. When the statements under the control of the PERFORM statement are executed the specified number of times, subscript-name-2 and subscript-name-3 have their initial values (FROM).

Figure 26. PERFORM Statement with Three Subscripts

Compiler Directing Verbs

Compiler directing verbs are special verbs that provide instructions for the COBOL compiler.

The EXIT verb is a dummy verb often used in the last statement in a series executed under the control of a PERFORM verb. The NOTE verb allows messages to be written during the execution of the program but does not affect the program in any other way. The STOP verb indicates the end of the program or a pause in the execution of the program. The ENTER verb is used to leave the main program and to transfer control to another program.

EXIT Verb

The EXIT statement provides a paragraph-name to serve as an end-point for a PERFORM statement when a series of procedures under the control of a PERFORM verb demand an ultimate common transfer point. In this case, an EXIT paragraph can be the object of GO TO statements in the series of statements under the control of a PERFORM statement. Its procedure-name may be given as the object of the THRU option in the PERFORM statement.

In all other cases, EXIT paragraphs perform no function and sequential control passes through them to the first sentence of the next paragraph. The format of the EXIT statement is:

```
paragraph-name. EXIT.
```

EXIT must appear as a single one-word paragraph.

NOTE Verb

The NOTE verb is used for writing explanatory statements that appear on the output listing but have no effect on the program. The format of the NOTE statement is:

```
NOTE any combination of characters from the COBOL character set
```

If NOTE is the first verb of a paragraph, the entire paragraph is treated as a note. If NOTE is not the first verb in a paragraph, the note ends with a period followed by a space.

STOP Verb

The STOP statement is used to halt the program temporarily or to indicate the end of a program or a program section. The format of the STOP statement is:

```
STOP { literal }
     { RUN     }
```

STOP LITERAL is usually specified for a temporary halt in the program to give instructions to the machine operator. The literal is displayed on the on-line printer. When the program is continued, the statement executed is the statement following the STOP verb.

STOP RUN is used to indicate the end of the program or the program section. This causes control to be returned to the system monitor or to a calling program that is using this program as a subprogram.

ENTER Verb

The ENTER verb provides for communication between a COBOL program and a subprogram that may be written in COBOL, FORTRAN IV (a scientific programming language), or MAP (a mnemonic programming language). In Part II of this publication, there are examples of communication between a COBOL program and subprograms written in COBOL, FORTRAN IV, and MAP. There is also an explanation of the use of the ENTER verb with FORTRAN IV mathematical subroutines from the library.

There are three forms of ENTER statements. The formats are:

```
CALL Form:
paragraph-name-1. ENTER LINKAGE-MODE.
  CALL 'entry-name' USING {data-name-1 ...}
                          {file-name-1 ... }
    [{data-name-m}] [data-name-m+1 ...]
     {file-name-m}                 [data-name-m+n]
    RETURNING procedure-name-1 [procedure-name-2 ...]
paragraph-name-2. ENTER COBOL.
```

```
ENTRY POINT Form:
paragraph-name-1. ENTER LINKAGE-MODE.
  ENTRY POINT IS 'entry-name'
                  [RECEIVE [data-name-1 ...] data-name-m]
            [PROVIDE [data-name-m+1 ...] date-name-m+n]
paragraph-name-2. ENTER COBOL.
```

```
RETURN Form:
paragraph-name-1. ENTER LINKAGE-MODE.
  RETURN VIA 'entry-name'
                  [DEPENDING ON data-name].
paragraph-name-2. ENTER COBOL.
```

The CALL form is used in the COBOL main program to transfer to the subprogram. *'Entry-name'* is the name of the point in the subprogram to which control is transferred. The USING option specifies the parameters that are passed between the main program and the subprogram. *Data-name-1* through *data-name-m* (*file-name-1* through *file-name-m*) name the data-items that are passed to the subprogram. *Data-name-m + 1* through *data-name-m + n* name data-items that receive data from the subprogram. The RETURNING option is used with the RETURN form of the ENTER statement to

specify the point in the main program to which control is returned after the subprogram is executed.

The ENTRY POINT form identifies the place in the subprogram to which control is transferred. *'Entry-name'* is the identifying name. This is the same *'entry-name'* that appears in the CALL form of the ENTER statement. The RECEIVE option specifies the data-items that receive data from the main program. The PROVIDE option specifies the data-items that return data to the main program.

The CALL form in the main program and the ENTRY POINT form in the subprogram must both be used in order to pass parameters from one COBOL program to the other. The data-items specified in the CALL form are matched to the data-items specified in the ENTRY POINT form by their relative position. The names need not be the same. The following example shows how the values are passed between programs.

Parameters Specified:

MAIN PROGRAM	SUBPROGRAM
CALL Form	ENTRY POINT Form
USING A, B, C, D	RECEIVE E, F PROVIDE G, H

Effect: When control is transferred to the subprogram, data-items E and F in the subprogram are set equal to the current value of data-items A and B in the main program. After the specified section subprogram is executed and control is returned to the main program, data-items C and D in the main program are set to the current value of data-items G and H in the subprogram.

The RETURN form identifies the end of the section of the subprogram to be executed and provides for return to the main program. *'Entry-name'* is the identifying name of the section of the subprogram to be executed. This is the same *'entry-name'* that appears in the CALL form and the ENTRY POINT form of the associated ENTER statements. Usually control is returned to the main program to the statement following the last statement of the CALL form, that is, to the statement after ENTER COBOL. The DEPENDING ON option in the RETURN form can be used with the RETURNING option in the CALL form to specify an alternate return. The return is to the 1st, 2nd, . . . , nth *procedure-name* following RETURNING as the value of *data-name* (following DEPENDING ON) is 1, 2, . . . , n. If the value of *data-name* is zero, the normal return is taken.

The following coding shows the use of the ENTER verb in a COBOL main program and a COBOL subroutine.

Main Program:

```
PARAGRAPH-NAME-1. ENTER LINKAGE-MODE.
    CALL 'ENTPT' USING WORK, WORK, AMOUNT.
PARAGRAPH-NAME-2. ENTER COBOL.
```

Subprogram:

```
PARAGRAPH-NAME-3. ENTER LINKAGE MODE.
    ENTRY POINT IS 'ENTPT'
    RECEIVE WORK-A PROVIDE WORK-A, AMOUNT-A.
PARAGRAPH-NAME-4. ENTER COBOL.
        .
        .
PARAGRAPH-NAME-5. ENTER LINKAGE-MODE.
    RETURN VIA 'ENTPT'.
PARAGRAPH-NAME-6. ENTER COBOL.
```

When control transfers to the subroutine, the value of WORK-A becomes equal to the value of WORK. Following PARAGRAPH-NAME-4 are COBOL procedure statements that affect the values of WORK-A and AMOUNT-A. After these statements are executed and PARAGRAPH-NAME-5 is encountered, control returns to the main program (to the statement after PARAGRAPH-NAME-2), and the values of WORK and AMOUNT in the main program are set equal to the values of WORK-A and AMOUNT-A at the end of the subroutine.

The following list summarizes the rules for using ENTER statements.

1. ENTER statements can be used only in COBOL programs. When one of the programs is written in another programming language, the statements from the language must be used in the program. Examples are given in Part II of this publication.

2. There may be more than one entry point in a subprogram. The *'entry-name'* identifies the point to which control is transferred.

3. *'Entry-name'* is the identifying name of the section of the subprogram to be executed. The same name appears in the three related ENTER statements.

4. *'Entry-name'* must consist of six or fewer characters chosen from the letters A through Z, the numbers 0 through 9, and the hyphen. There must be at least one alphabetic character, and there may be no embedded blanks. The name must be enclosed in quotation marks.

5. When the ENTRY POINT form is used, *'entry-name'* cannot be the deckname of the subprogram.

6. If the *'entry-name'* in the CALL form of the ENTER statement of the main program is the deckname of the subprogram (on the $IBCBC control card), the RETURN form need not be used in the subprogram. The statement STOP RUN causes control to be returned to the main program.

7. The parameters specified in the CALL form and the ENTRY POINT form to pass data between the programs must conform to the following rules.

   a. The number of data-names following USING in the CALL form must equal the sum of the data-names following RECEIVE and PROVIDE in the ENTRY POINT form.

b. The data-items are matched by their relative positions. The names following USING in the CALL form need not be identical to the names in the corresponding positions following RECEIVE and PROVIDE in the ENTRY POINT form, but the data-item descriptions must be equivalent.

c. Data-names may be repeated.

d. The data-items must not be variable in length.

e. Data-items following RECEIVE and PROVIDE in the ENTRY POINT form may not be report items; however, scientific decimal items are permitted. Although report items may follow USING in the CALL form, the restriction for the ENTRY POINT form requires that the subprogram be written in a language other than COBOL.

f. File-names can follow the USING option only if the subprogram is written in the MAP language. The address of the corresponding file control block is provided to the MAP program. File control blocks are explained in the publication *IBM 7090/7094 Input/Output Control System*, Form C28-6392.

Tests — Conditional Expressions

A conditional expression is one that may be either true or false when it is evaluated. For example, the expression YEARLY-INCOME IS GREATER THAN 5000 is a conditional expression, since it is true or false depending on the value of YEARLY-INCOME when the statement is executed.

Conditional statements, that is, those containing conditional expressions, are of the following forms:

Option 1.

```
IF conditional-expression    statement-1
```

Option 2.

```
IF conditional-expression { statement-1      }
                          { NEXT SENTENCE    }
  { OTHERWISE }   { statement-2      }
  { ELSE      }   { NEXT SENTENCE    }
```

In option 1, if the condition is met, the next statement to be executed is *statement-1*. If the condition is not met, the next statement to be executed is the statement following the conditional statement.

In option 2, if both *statement-1* and *statement-2* are specified, *statement-1* is executed when the condition is met, and *statement-2* is executed when the condition is not met. If NEXT SENTENCE and *statement-2* are specified in that order, *statement-2* is executed only when the condition is not met; otherwise, the next sequential instruction is executed. Specifying *statement-1* and NEXT SENTENCE in that order is equivalent to using option 1.

The ON SIZE ERROR clause of arithmetic statements, the AT END clause of the READ statement, and the GO TO DEPENDING ON statement are special kinds of conditional expressions. In these statements, the ELSE (OTHERWISE) option is not permitted.

Simple Conditional Tests

There are five types of simple conditional tests. In each, the test is to determine if the specified condition is met.

Relation Test

A relation test is a comparison of the size of two operands. The format is:

```
IF { data-name-1          }  { IS [NOT] GREATER THAN }
   { literal-1            }  { IS [NOT] LESS THAN    }
   { formula-1            }  { IS [NOT] EQUAL TO     }
   { figurative constant-1}  {     [NOT] =           }

                                { data-name-2          }
                                { literal-2            }
                                { formula-2            }
                                { figurative constant-2}
```

If both of the operands are numeric, the algebraic values are compared. If one or both of the operands are nonnumeric, they are compared with respect to a collating sequence. (The commercial collating sequence is assumed unless the 7090/7094 collating sequence is specified by placing BINSEQ on the $IBCBC control card; these collating sequences are shown in "Appendix B.") The operands are compared from left to right. If they are of unequal length, the longer is assumed to be the greater; however, leading zeros and blanks and trailing blanks are ignored.

The operands cannot be both literals or figurative constants.

Sign Test

The sign test determines whether the value of a data-item or a formula is greater than, less than, or equal to zero. The format is:

```
IF { data-name } IS [NOT] { POSITIVE }
   { formula   }          { NEGATIVE }
                          { ZERO     }
```

Class Test

The class test determines whether data is alphabetic or numeric. The test can be made only for an ALPHANUMERIC data-item. The format is:

```
IF data-name IS [NOT] { NUMERIC    }
                      { ALPHABETIC }
```

A NUMERIC data-item consists of characters chosen from the numbers 0 through 9 and the operational signs. An ALPHABETIC data-item consists of characters chosen from the letters A through Z and the space. When a single-character item is an operational sign, it is considered NUMERIC if the test is for a NUMERIC item and ALPHABETIC if the test is for an ALPHABETIC item.

Conditional Variable Test

A condition-name is a name assigned to one of the values in the range of values a data-item can assume. Condition-names are identified by the special level-number 88; an example is given in the description of the VALUE clause of the data-item description in the FILE SECTION. The data-item that assumes the range of values is called a conditional variable. The conditional variable test determines whether the value of the conditional variable is equal to the value of a condition-name associated with it. The format is:

```
IF [NOT] condition-name
```

Switch Status Test

The OFF position and the ON position of the entry keys on the computer may be assigned switch-status names in the SPECIAL-NAMES paragraph in the ENVIRONMENT DIVISION. The switch status test provides a means of determining whether an entry key is on or off. The format is:

```
IF [NOT] switch-status-name
```

Compound Conditions

Simple conditional tests are for a single condition. The tests may be combined with the logical connectives AND and OR to form compound conditions. Figure 27 shows the effect of the logical connectives on simple conditions C1 and C2.

C1	C2	C1 AND C2	C1 OR C2
True	True	True	True
False	True	False	True
True	False	False	True
False	False	False	False

Figure 27. Logical Connectives

The expression C1 AND C2 is equivalent to the expression "both C1 and C2," that is, the condition is true only if C1 is true and C2 is true. The expression C1 OR C2 is equivalent to the expression "either C1 or C2," that is, the condition is true if C1 is true, or if C2 is true, or if both C1 and C2 are true.

Any number of operands may be combined according to these rules. If both logical connectives, AND and OR, are used in an expression, parentheses may be used to indicate grouping. The parentheses must always be paired; and the contents of the innermost parentheses are evaluated first as in algebra. If parentheses are not used, then the conditions are grouped first according to AND, proceeding from left to right, and then by OR, proceeding from left to right.

The following examples show how the compiler evaluates compound conditions when C1 and C2 are true conditions and C3 and C4 are false conditions.

Example 1:

```
Evaluate:   C1 AND (C2 OR NOT (C3 OR C4)).
Solution:   true AND (true OR NOT (false OR false)).
            = true AND (true OR NOT false).
            = true AND (true OR true).
            = true AND true.
            = true.
```

Example 2:

```
Evaluate:   C1 OR C2 AND C3.
Solution:   true OR true AND false.
            = true OR (true AND false).
            = true OR false.
            = true.
```

Example 3:

```
Evaluate:   (C1 AND C2) OR (NOT C3 AND C4).
Solution:   true AND true OR NOT false AND false.
            = (true AND true) OR (NOT false AND false).
            = true OR (true AND false).
            = true OR false.
            = true.
```

The logical operators AND and OR may also be used to combine operands. The effect is the same as when they are used to combine simple conditions. For example, the expression IF SUM AND DIFFERENCE POSITIVE is equivalent to the expression IF SUM POSITIVE AND IF DIFFERENCE POSITIVE.

Nested Conditionals

In option 2 of the format of a conditional statement; that is,

```
IF conditional-expression { statement-1     }
                          { NEXT SENTENCE   }

   { OTHERWISE } { statement-2     }
   { ELSE      } { NEXT SENTENCE   }
```

statement-1 and *statement-2* may consist of one or more imperative statements and/or a conditional statement. If a conditional statement appears as *statement-1* or as part of *statement-1*, it is said to be nested. A nested conditional statement is completely contained in another statement. The IF and ELSE combinations are paired from the inside outward and the sentence is evaluated from left to right. Figure 28 shows the evaluation of a nested conditional statement.

Statement:

```
IF PRINCIPAL LESS TAX ADD 1 TO COUNT IF PRINCIPAL LESS INSUR-
ANCE MOVE CORRESPONDING A TO B  IF COUNT GREATER 10 GO
TO ERR ELSE PERFORM PARAG1 THRU PARAG6 ELSE MOVE CORRESPOND-
ING A TO C ELSE COMPUTE RATE = INTEREST / (PRINCIPAL * TIME).
```

Evaluation:

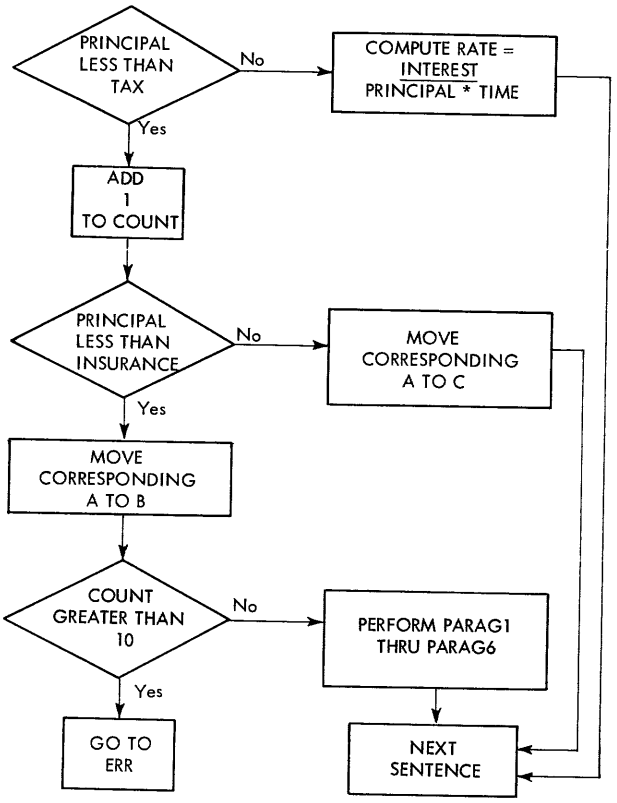


Figure 28. Evaluation of a Nested Conditional Statement

ENTER Verb

The following examples illustrate the use of the ENTER verb. The rules for the ENTER verb were given in Part I of this publication, but the formats are repeated here for ease of reference.

```
CALL Form:
paragraph-name-1.  ENTER LINKAGE-MODE.
  CALL  'entry-name' [ USING {data-name-1 ...}
                             {file-name-1 ...} ]
[ {data-name-m} ] [data-name-m+1 ...] [data-name-m+n]
  {file-name-m}
      RETURNING procedure-name-1 [procedure-name-2 ...]
paragraph-name-2.  ENTER COBOL.
```

```
ENTRY POINT Form:
paragraph-name-1.  ENTER LINKAGE-MODE.
  ENTRY POINT IS 'entry-name'
                [ RECEIVE [data-name-1 ...] data-name-m ]
              [ PROVIDE [data-name-m+1 ...] data-name-m+n ]
paragraph-name-2.  ENTER COBOL.
```

```
RETURN Form:
paragraph-name-1.  ENTER LINKAGE-MODE.
  RETURN VIA 'entry-name' [DEPENDING ON data-name.]
paragraph-name-2.  ENTER COBOL.
```

FORTRAN IV Library Subroutines

The FORTRAN IV mathematical library subroutines can be used in a COBOL program without requiring the use of a programming language other than COBOL. Thirty-two mathematical functions can be evaluated with these subroutines. The evaluation of the functions is actually performed by the 7090/7094 FORTRAN IV mathematics library, but the linkage to the subroutines (FORTRAN-ACCESS feature of COBOL) is in the COBOL language. All data-items used as operands or results are defined as normal COBOL data-items that may be manipulated in the COBOL language. The variables to be used in the computations are supplied to the subroutine, the computation is performed, and the result is returned to the COBOL program. Special COBOL-names have been assigned to the entry points of each subroutine. The general format for calling the FORTRAN IV library subroutines is:

```
paragraph-name-1.  ENTER LINKAGE-MODE.
  CALL 'COBOL-name'  USING data-name-1 data-name-2
                                       [data-name-3].
paragraph-name-2.  ENTER COBOL.
```

The result of the calculation is stored in *data-name-1*. *Data-name-2*, and *data-name-3* if applicable, are the operands. Figure 29 shows the types of operations that can be performed, the COBOL-names for the subroutines, and the types of data that can be used for each subroutine.

Types of Data

The following list shows the type of data that may be used as operands in the FORTRAN IV library subroutines. The data types correspond to the numbers in parentheses following the operands in Figure 29.

The complete range of values for the type of number is given in the list. However, the acceptable range of the variables for each subroutine is determined by the subroutine, and the COBOL compiler does not check their validity. The accuracy of the COBOL results is approximately the same as the FORTRAN IV subroutine accuracy. The subroutines also include error procedures. The FORTRAN IV subroutines are completely documented in the library section of the publication *IBM 7090/7094 IBSYS Operating System, IBJOB Processor*, Form C28-6389.

1. *Single-Precision Fixed-Point:*
   Values data-item may assume: Integers of 10 or fewer digits.

```
level-number data-name PICTURE 9(10)
  SYNCHRONIZED RIGHT USAGE COMPUTATIONAL
                [VALUE numeric-literal] [OCCURS ...]
```

NOTE: The item must be an integer (no V in the PICTURE clause) and the size must be given as 10.

2. *Double-Precision Fixed-Point:*
   Values data-item may assume: Integers of 20 or

C = f(A, B) where C is the result, and A and B are the operands.
(The numbers in parentheses are data types.)

f(A, B)	FORTRAN Name	COBOL Name	Result	Operands	
			data-name-1	data-name-2	data-name-3
A^B	.XP1.	.CXP1 *	C(1)	A(1)	B(1)
A^B	.XP2.	.CXP2 *	C(3)	A(3)	B(1)
A^B	.XP3.	.CXP3 *	C(3)	A(3)	B(3)
A^B	.DXP1.	.CDXP1 *	C(4)	A(4)	B(1)
A^B	.DXP2.	.CDXP2 *	C(4)	A(4)	B(4)
A^B	.CXP1.	.CCXP1	C(5)	A(5)	B(1)
e^A	EXP	.CEXP	C(3)	A(3)	
e^A	DEXP	.CDEXP	C(4)	A(4)	
e^A	CEXP	.CCEXP	C(5)	A(5)	
ln(A)	ALOG	.CALOG	C(3)	A(3)	
ln(A)	DLOG	.CDLOG	C(4)	A(4)	
ln(A)	CLOG	.CCLOG	C(5)	A(5)	
$\log_{10}(A)$	ALOG10	.CAL10	C(3)	A(3)	
$\log_{10}(A)$	DLOG10	.CDL10	C(4)	A(4)	
$\text{Tan}^{-1}(A)$	ATAN	.CATAN	C(3)	A(3)	
$\text{Tan}^{-1}(A)$	DATAN	.CDATN	C(4)	A(4)	
$\text{Tan}^{-1}(A/B)$	ATAN2	.CATN2	C(3)	A(3)	B(3)
$\text{Tan}^{-1}(A/B)$	DATAN2	.CDAT2	C(4)	A(4)	B(4)
Sin(A)	SIN	.CSIN	C(3)	A(3)	
Sin(A)	DSIN	.CDSIN	C(4)	A(4)	
Sin(A)	CSIN	.CCSIN	C(5)	A(5)	
Cos(A)	COS	.CCOS	C(3)	A(3)	
Cos(A)	DCOS	.CDCOS	C(4)	A(4)	
Cos(A)	CCOS	.CCCOS	C(5)	A(5)	
Tanh(A)	TANH	.CTANH	C(3)	A(3)	
$\sqrt{A}$	SQRT	.CSQRT	C(3)	A(3)	
$\sqrt{A}$	DSQRT	.CDSQR	C(4)	A(4)	
$\sqrt{A}$	CSQRT	.CCSQR	C(5)	A(5)	
A MOD B	DMOD	.CDMOD	C(4)	A(4)	B(4)
$\lvert A \rvert$	.CABS.	.CCABS	C(3)	A(5)	
A*B	.CFMP.	.CCFMP	C(5)	A(5)	B(5)
A/B	.CFDP.	.CCFDP	C(5)	A(5)	B(5)

*Note the COBOL language already provides exponentiation (except complex) in the COMPUTE statement.

Figure 29. FORTRAN IV Mathematical Subroutines

fewer digits. This type of item may not be used with the subroutines at the present time.

3. *Single-Precision Floating-Point:*
Values data-item may assume: Real numbers of the form a * 10^b where a is a real number of 8 or fewer digits and b ranges from −38 to +38.

level-number data-name USAGE COMPUTATIONAL-1
[VALUE { numeric-literal / floating-point-literal }] [OCCURS ...].

4. *Double-Precision Floating-Point:*
Values data-item may assume: Real numbers of the form a * 10^b where a is a real number of 16 or fewer digits and b ranges from −38 to +38.

level-number data-name USAGE COMPUTATIONAL-2
[VALUE { numeric-literal / floating-point-literal }] [OCCURS ...].

5. *Complex Numbers:* A complex number has the form x + iy where i is the imaginary number $\sqrt{-1}$. Since COBOL normally has no convention for handling complex numbers, the following rules have been established:

a. A complex number is defined as a group item containing two elementary items that are described as single-precision floating-point (item 3) items.

b. The CORRESPONDING option of the ADD, SUBTRACT, and MOVE statements may be used to facilitate the handling of complex data-items. When this is done, a naming convention must be followed that conforms to the rules for the CORRESPONDING option.

c. Tables of complex numbers may be set up by using the OCCURS clause at the group level. The OCCURS clause must not be used for the elementary items, but otherwise normal rules of the OCCURS clause apply.

d. When a complex number is displayed, the DISPLAY statement must refer to the elementary items.

The following example shows typical entries required to perform the addition (1 + i2) + (7 + i4) = 8 + i6. REAL and IMAG are used as the naming convention in the example.

```
DATA DIVISION Entries:
  01 A.
     02 REAL USAGE IS COMPUTATIONAL-1
        VALUE IS 1.
     02 IMAG USAGE IS COMPUTATIONAL-1
        VALUE IS 2.
  01 B.
     02 REAL USAGE IS COMPUTATIONAL-1
        VALUE IS 7.
     02 IMAG USAGE IS COMPUTATIONAL-1
        VALUE IS 4.
PROCEDURE DIVISION Entries:
  ADD CORRESPONDING A TO B.
```

The elementary items may also be referred to individually, as in the following statements.

```
ADD REAL OF A TO REAL OF B.
MOVE 3 TO IMAG OF B.
DISPLAY REAL OF A, SPACE, IMAG OF A.
```

Example of Use of FORTRAN IV Library Subroutines

Summary of the Problem: This program shows how the FORTRAN IV mathematical subroutines can be used by a COBOL program to evaluate a mathematical formula. The subroutines .CCEXP (natural exponentiation) and .CCFMP (complex multiplication) are used. The problem is to evaluate the formula:

$$y = ae^{jw(t-\frac{x}{c})}$$

Step 1. Use COBOL to calculate the value of the exponent

$$w(t-\frac{x}{c})$$

Step 2. Use the FORTRAN IV access subroutine .CCEXP to calculate

$$e^{jw(t-\frac{x}{c})}$$

(natural complex exponentiation)

Step 3. Use the FORTRAN IV access subroutine .CCFMP to calculate

$$ae^{jw(t-\frac{x}{c})}$$

(complex multiplication)

Step 4. Use COBOL to edit the results and display them.

Program Listing of FORTRAN IV Library Subroutines:

```
$JOB                    CALCULATION OF HARMONIC
$*                        WAVE AMPLITUDE
$EXECUTE                IBJOB
$IBJOB      FTC         GO, MAP
$IBCBC      WAVE        FULIST, NODECK
            IDENTIFICATION DIVISION.
            PROGRAM-ID. HARMONIC-WAVE-
              AMPLITUDE.
            REMARKS. THE GENERAL EXPRESSION FOR
              A HARMONIC WAVE TRAVELLING TO THE
              RIGHT (I.E., POSITIVE X) IN AN IDEAL
              MEDIUM IS . . .
                  Y = A * (E ** (J * W * (T - X / C)))
              WHERE
                Y IS THE AMPLITUDE OF THE WAVE X
                  AT TIME T.
                E IS THE NATURAL BASE.
                J IS THE IMAGINARY NUMBER.
                W IS THE FREQUENCY, ASSUMED
                  KNOWN AND CONSTANT.
                T IS THE TIME.
                X IS THE DISPLACEMENT, ASSUMED
                  KNOWN AND CONSTANT.
                C IS THE VELOCITY OF WAVE
                  PROPAGATION, ASSUMED KNOWN
                  AND CONSTANT.
                A IS A CONSTANT, ASSUMED KNOWN.
              BOTH Y AND A ARE COMPLEX NUMBERS.
                A IS PRESUMED TO HAVE A NON-ZERO
                IMAGINARY PART.
              PROBLEM . . .
                TO LIST, FOR TIMES FROM 0 TO 100,
                THE DISPLACEMENT OF Y. (THIS MAY
                HAVE NON-ZERO IMAGINARY PART
                DUE TO ASSUMPTIONS MADE FOR A).
            DATA DIVISION.
            WORKING-STORAGE SECTION.
            01 T PICTURE 9(3) COMPUTATIONAL
               SYNCHRONIZED RIGHT.
            01 C COMPUTATIONAL-1 VALUE 30.
            01 X COMPUTATIONAL-1 VALUE 30.
            01 W COMPUTATIONAL-1 VALUE 25.
            01 EXP-COMPLEX COMPUTATIONAL-1.
               02 EXP-REAL.
               02 EXP-IMAG.
            01 E-TERM COMPUTATIONAL-1.
               02 E-REAL.
               02 E-IMAG.
            01 A-COMPLEX COMPUTATIONAL-1.
               02 A-REAL VALUE 500.
               02 A-IMAG VALUE 325.
            01 Y-COMPLEX COMPUTATIONAL-1.
               02 Y-REAL.
               02 Y-IMAG.
            01 SHOW-REAL PICTURE XXXXXX.
            01 SHOW-IMAG PICTURE XXXXXX.
            PROCEDURE DIVISION.
            GENERAL SECTION.
            G0. DISPLAY 'COMPLEX AMPLITUDES OF A
                HARMONIC WAVE AT A POINT X'
                'FROM T = 0 TO T = 100.' UPON SYSOU1.
            G1. PERFORM Y-LOOP VARYING T FROM 0
                BY 1 UNTIL T = 101.
            G2. STOP RUN.
```

```
Y-LOOP SECTION.
Y1. COMPUTE EXP-IMAG = W * (T - X / C)
    ON SIZE ERROR MOVE ZERO TO EXP-IMAG.
Y2. MOVE ZERO TO EXP-REAL.
Y3. ENTER LINKAGE-MODE.
    CALL '.CCEXP' USING E-TERM, EXP-
        COMPLEX.
    CALL '.CCFMP' USING Y-COMPLEX,
        A-COMPLEX, E-TERM.
Y4. ENTER COBOL.
    MOVE Y-REAL TO SHOW-REAL.
    MOVE Y-IMAG TO SHOW-IMAG.
Y5. DISPLAY ' AT TIME ', T, ' DISPLACEMENT
        IS ', SHOW-REAL, ' + J * ', SHOW-IMAG
        UPON SYSOU1.
$CBEND
```

Main Program Language	Statements Used	Subprogram Language	Statements Used
COBOL	CALL form of ENTER statement	COBOL	ENTRY POINT and RETURN forms of ENTER statement
COBOL	CALL form of ENTER statement	MAP	SAVE AND RETURN
COBOL	CALL form of ENTER statement	FORTRAN IV	SUBROUTINE and RETURN
MAP	CALL	COBOL	ENTRY POINT and RETURN forms of ENTER statement
FORTRAN IV	CALL	COBOL	ENTRY POINT and RETURN forms of ENTER statement

Figure 30. Statements Used in COBOL, MAP, and FORTRAN IV to Enter the Linkage Mode

COBOL Linkage to MAP, FORTRAN IV, and COBOL Subprograms

The ENTER verb allows control to be transferred and data to be passed between a COBOL program and subprograms written in MAP, FORTRAN IV, or COBOL.

The types of data acceptable for FORTRAN IV mathematical library subroutines are also acceptable to subprograms written in MAP or FORTRAN IV. These types are listed under the discussion of the FORTRAN IV mathematical library subroutines. The general restrictions on data passed between programs apply for linkage to programs written in COBOL, MAP, or FORTRAN IV. The accumulator and other registers may not be used as parameters in linkage between a COBOL program and any subprogram since the COBOL compiler generates coding that destroys the contents of these registers. The instructions for entering the linkage mode are written in the language of the program or subprogram. Figure 30 shows instructions for entering the linkage mode for COBOL, FORTRAN IV, and MAP. Detailed rules for using FORTRAN IV and MAP linkage mode statements are given under the names of the statements in the publications: *IBM 7090/7094 IBSYS Operating System: FORTRAN IV Language,* Form C28-6390, and *IBM 7090/7094 IBSYS Operating System: Macro Assembly Program (MAP) Language,* Form C28-6392.

Summary of Problem: This example shows the linkage between a COBOL program and a COBOL subprogram, a MAP subprogram, and a FORTRAN IV subprogram.

The COBOL main program reads payroll cards and calls a COBOL subprogram to calculate regular pay and return this information to the main program. The return to the main program is to one of two subroutines depending on whether the employee is exempt or non-exempt from overtime pay. The non-exempt subroutine calls a FORTRAN IV subprogram that calculates overtime pay (at the rate of time and a half) and returns the information to the main program. The main program then calls a MAP subroutine that calculates double-time pay and returns the information to the main program. Then the total pay for both exempt and non-exempt employees is displayed.

Program Listing of COBOL, MAP, and FORTRAN IV Linkage:

COBOL Main Program

```
$JOB                    PAYROLL CALCULATION TO
$*                        GROSS
$EXECUTE                IBJOB
$IBJOB          PAY     GO, MAP
$IBCBC          CBC1    FULIST, NODECK
                IDENTIFICATION DIVISION.
                PROGRAM-ID. PAY-CALCULATION-TO-GROSS.
                AUTHOR. ACCOUNTING MANAGER D. J. M.
                INSTALLATION. PAYROLL DEPARTMENT.
                DATE-WRITTEN. 7/21/64.
                REMARKS. THIS ROUTINE DEMONSTRATES
                    THE CALLING OF COBOL, FORTRAN,
                    AND MAP PROGRAMS FROM COBOL.
                DATA DIVISION.
                WORKING-STORAGE SECTION.
                77 O-T-COMP PICTURE 9(8)V99
                    COMPUTATIONAL SYNCHRONIZED
                    RIGHT.
                77 D-T-COMP PICTURE 9(8)V99
                    COMPUTATIONAL SYNCHRONIZED
                    RIGHT.
                77 RATE-COMP PICTURE 9(7)V999
                    COMPUTATIONAL SYNCHRONIZED
                    RIGHT.
                01 HRS-CARD.
                    02 MAN-NO PICTURE 9(6).
```

```
      02 REG-HRS PICTURE 9999V99.
      02 O-T-HRS PICTURE 9999V99.
      02 D-T-HRS PICTURE 9999V99.
   01 RATE PICTURE 999V999.
   01 PAY-SUMS, USAGE IS COMPUTATIONAL.
      02 STRAIGHT-PAY PICTURE 9(7)V999
         SYNCHRONIZED RIGHT.
      02 O-T-PAY PICTURE 9(7)V999
         SYNCHRONIZED RIGHT.
      02 D-T-PAY PICTURE 9(7)V999
         SYNCHRONIZED RIGHT.
   01 GROSS PICTURE $$$$$.99.
   01 NAME-B.
      02 NAME PICTURE X(18).
      02 B PICTURE X VALUE SPACE.
   PROCEDURE DIVISION.
   GROSS-PAY-REPORT. ACCEPT HRS-CARD FROM
         SYSIN1.
      IF HRS-CARD = ZERO STOP RUN.
      MOVE 0 TO O-T-PAY, D-T-PAY.
   STRAIGHT-PAY-CALC.
      ENTER LINKAGE-MODE.
      CALL 'COBOLS' USING MAN-NO, REG-HRS,
         NAME, RATE, STRAIGHT-PAY,
         RETURNING EXEMPT, NON-EXEMPT.
   RETURN-TO-COBOL-1. ENTER COBOL.
   NON-EXEMPT.
   BCD-TO-BIN-CONVERSION.
      MOVE O-T-HRS TO O-T-COMP.
      MOVE D-T-HRS TO D-T-COMP.
      MOVE RATE TO RATE-COMP.
   O-T-CALC.
      IF O-T-HRS = ZERO GO TO D-T-CALC.
   O-T-CALC-BY-FORTRAN.
      ENTER LINKAGE-MODE.
      CALL 'FORTRS' USING O-T-COMP,
            RATE-COMP, O-T-PAY.
   RETURN-TO-COBOL-2. ENTER COBOL.
   D-T-CALC. IF D-T-HRS = 0 GO TO CALC-GROSS.
   D-T-CALC-BY-MAP. ENTER LINKAGE-MODE.
      CALL 'MAPS' USING D-T-COMP,
            RATE-COMP, D-T-PAY.
   RETURN-TO-COBOL-3. ENTER COBOL.
   EXEMPT.
   CALC-GROSS. COMPUTE GROSS =
         STRAIGHT-PAY + O-T-PAY + D-T-PAY.
   SHOW-GROSS-PAY-REPORT.
      DISPLAY NAME-B, MAN-NO, B GROSS
            UPON SYSOU1.
      GO TO GROSS-PAY-REPORT.
$CBEND
```

COBOL Subprogram

```
$IBCBC      CBC2        FULIST, NODECK
   IDENTIFICATION DIVISION.
   PROGRAM-ID. COBOLS.
   AUTHOR. D. J. M.
   INSTALLATION. PAYROLL DEPARTMENT.
   DATE WRITTEN. 7/21/64.
   REMARKS.
      FIND NAME, RATE, AND CALCULATE
         STRAIGHT PAY.
   ENVIRONMENT DIVISION.
   INPUT-OUTPUT SECTION.
   FILE-CONTROL.
      SELECT EMPLOYEE-RECORDS ASSIGN TO
         A(1).
   DATA DIVISION.
   FILE SECTION.
   FD EMPLOYEE-RECORDS LABEL RECORDS
         ARE OMITTED.
      DATA RECORD IS TBL-AREA.
   01 TBL-AREA.
      02 EMPLOYEE-COUNT PICTURE 9(4).
      02 MAN-RECORD OCCURS 1000 TIMES
            DEPENDING ON EMPLOYEE-COUNT.
         03 MAN-NO PICTURE 9(6).
         03 HOURLY-RATE PICTURE 999V999.
         03 X PICTURE X(6).
         03 MAN-NAME PICTURE X(18).
   WORKING-STORAGE SECTION.
   77 ZRO COMPUTATIONAL SYNCHRONIZED
         RIGHT PICTURE 9(10), VALUE 0.
   77 DIRECTION PICTURE 99999
         COMPUTATIONAL, SYNCHRONIZED
         RIGHT.
   01 EMPLOYEE-NUMBER PICTURE 9(6).
   01 NAME-OF-EMP PICTURE X(18).
   01 RATE PICTURE 999V999.
   01 REG-HRS PICTURE 999V99.
   01 STRAIGHT-PAY PICTURE 999V999
         COMPUTATIONAL SYNCHRONIZED
         RIGHT.
   PROCEDURE DIVISION.
   ENTRY-TO-LOCATE-RATE.
      ENTER LINKAGE-MODE.
         ENTRY POINT IS 'COBOLS' RECEIVE
            EMPLOYEE-NUMBER, REG-HRS,
            PROVIDE NAME-OF-EMP, RATE,
            STRAIGHT-PAY.
   RETURN-TO-COBOL-A. ENTER COBOL.
   PIVOT-POINT. GO TO FIRST-TIME-THRU.
   FIRST-TIME-THRU.
      OPEN INPUT EMPLOYEE-RECORDS.
      READ EMPLOYEE-RECORDS AT END NEXT
         SENTENCE.
      ALTER PIVOT-POINT TO PROCEED TO
         ALL-TIMES-THRU.
   ALL-TIMES-THRU.
      PERFORM FIND-NAME-RATE VARYING
```

```
            TALLY FROM 1 BY 1 UNTIL TALLY
            IS GREATER THAN EMPLOYEE-COUNT.
        NOT-FOUND. DISPLAY 'MAN NUMBER'
            EMPLOYEE-NUMBER 'UNASSIGNED.'
            UPON SYSOU1.
          MOVE 'UNKNOWN' TO NAME-OF-EMP,
            MOVE 0 TO RATE, STRAIGHT-PAY,
            MOVE 1 TO DIRECTION, GO TO
            CONCLUSION.
        FOUND. MOVE MAN-NAME (TALLY) TO
          NAME-OF-EMP, MOVE HOURLY-RATE
          (TALLY) TO RATE.
        CALC-STRAIGHT-PAY. IF X (TALLY) = 'E',
          MOVE RATE TO STRAIGHT-PAY, MOVE 1
          TO DIRECTION, GO TO CONCLUSION. COM-
          PUTE STRAIGHT-PAY = REG-HRS * RATE.
          MOVE 2 TO DIRECTION.
        CONCLUSION. ENTER LINKAGE-MODE.
          RETURN VIA 'COBOLS' DEPENDING ON
          DIRECTION.
        RETURN-TO-COBOL-B. ENTER COBOL.
          CLOSE EMPLOYEE-RECORDS.
        FIND-NAME-RATE SECTION.
          IF EMPLOYEE-NUMBER = MAN-NO
          (TALLY) GO TO FOUND.
$CBEND
```

FORTRAN IV Subprogram

```
$IBFTC    FTC1      FULIST, NODECK
          SUBROUTINE FORTRS (IOTHRS,IRATE,IOTPAY)
          IOTPAY = (IOTHRS*IRATE*15)/1000
          RETURN
          END
```

MAP Subprogram

```
$IBMAP    MAP1      FULIST, NODECK
          ENTRY     MAPS
MAPS      SAVE      4         DOUBLE-TIME CALCUL-
          LDQ*      3,4       ATION . . . . .
          MPY*      4,4         DOUBLE-TIME-HRS
*                                 IN MQ
*                             MULTIPLY D-T-HRS
*                               BY RATE, RESULT
          DVP       =50         IN AC, MQ WITH
          STQ*      5,4         5 DECIMAL PLACES
          RETURN    MAPS      THE C(AC, MQ) ARE
          END                   TO BE MULTIPLIED
*                               BY 2 AND DOWN-
*                               SCALED 2 PLACES
*                               TO GIVE 2 DEC.
*                               PLACES.
*                           STORE DOUBLE-TIME-
*                               PAY INTO CALLING
*                               PROGRAM.
```

Dump Subroutine

The dump subroutine is used to obtain core storage dumps (selected portions of core storage written on magnetic tape). The format for calling the dump subroutine is:

```
paragraph-name-1. ENTER LINKAGE-MODE.
  CALL {'DUMP' }
       {'PDUMP'}
            [USING data-name-1 data-name-2 data-name-3]
                   [data-name-4 data-name-5 data-name-6].
paragraph-name-2. ENTER COBOL.
```

The entry point 'DUMP' is used to obtain a dump of selected portions of core storage and return control to the IBJOB Processor Monitor.

The entry point 'PDUMP' is used to obtain a dump of selected portions of core storage during the execution of the program. After the dump has been taken, execution of the program is resumed.

The parameters that follow USING specify the portions of core storage that are to be dumped, and the type of dump that is to be taken. The values stored from *data-name-1* (*data-name-4*, etc.) through *data-name-2* (*data-name-5*, etc.) are dumped. The data-names may be any names in the WORKING-STORAGE or CONSTANT SECTIONS or may be file-names. When a file-name is used as a parameter, the file control block is dumped.

Data-name-3 (*data-name-6*, etc.) must contain codes that specify the type of dump to be taken. The codes and their meanings are:

CODE	MEANING
0	octal
1	floating-point
2	integer
3	octal and mnemonics

If USING is omitted all of core storage is dumped in octal.

USE Verb

The USE verbs appear in the declaratives section of the PROCEDURE DIVISION and allow the user to specify routines to be executed at the time labels are written or checked and at the time input/output errors occur. The rules for the USE verb are given in Part I of this publication but the format is repeated here for easy reference.

```
Option 1.
section-name SECTION.
  USE AFTER STANDARD ERROR PROCEDURE
        ON {INPUT                          }
           {file-name-1 [file-name-2 . . .]} .
```

Option 2.
section-name SECTION.

```
USE {BEFORE} STANDARD [{BEGINNING}]
    {AFTER }          [{ENDING   }]
    [{REEL}]
    [{FILE}] LABEL PROCEDURE
                ON {INPUT                          }
                   {OUTPUT                         }
                   {file-name-1 [file-name-2 . . .]} .
```

Label Processing and Error Checking Routine

There are two USE statements in the declaratives section of this program. The first USE statement initiates procedures to alter a standard label before it is checked. Since a COBOL programmer cannot make reference to the label area, a MAP subprogram is called to move the standard label from the label area to a work area that can be referred to by the COBOL program. The COBOL program puts the customer's name in the label and calls a MAP subprogram to return the altered label to the label area. In the regular sequence of procedure statements (not in the declaratives section), the altered label is displayed.

The second USE statement specifies procedures to be followed if an input-output error occurs. Normally, if an error cannot be corrected, the input-output system puts codes indicating the type of error in the multiplier-quotient register (an internal register that cannot be referred to in a COBOL program) and transfers control to a COBOL compiler subroutine that prints messages indicating the type of error that has occurred, takes a core storage dump, and terminates execution. If USE statements are given, they replace the normal COBOL compiler subroutine. In this program, a MAP subprogram is called to store the error codes in a word that can be referred to by the COBOL program. During the regular sequence of execution statements (not in the declaratives section) this word is tested, and the type of error, if any, is displayed. In this program, no dump is taken and program execution is not terminated if an input-output error occurs.

Program Listing of Label Processing and Error Checking Routine:

COBOL Main Program

```
$JOB                    COBOL LABEL ACCESS AND ERROR
$*                            CHECKING DEMO PROGRAM.
$EXECUTE                IBJOB
$IBJOB      DEMO        GO, MAP
$IBCBC      USELBL      FULIST, NODECK
            ENVIRONMENT DIVISION.
            INPUT-OUTPUT SECTION.
            FILE-CONTROL.
              SELECT A-FILE ASSIGN TO A(1), A(2)
                    FOR MULTIPLE REEL.
DATA DIVISION.
FILE SECTION.
FD A-FILE LABEL RECORDS ARE STANDARD
     RECORDING MODE IS BINARY LOW,
     DATA RECORD IS SIMPLE.
01 SIMPLE PICTURE X(72).
WORKING-STORAGE SECTION.
01 SIZE-OF-LABEL-IN-WORDS PICTURE 9(10)
     COMPUTATIONAL VALUE 14.
01 STANDARD-LABEL.
   02 LABEL-ID              PICTURE X(6).
   02 FILLER                PICTURE X.
   02 TAPE-SER-NUMBER       PICTURE X(5).
   02 FILLER                PICTURE X.
   02 FILE-SER-NUMBER       PICTURE X(5).
   02 FILLER                PICTURE X.
   02 REEL-SEQ-NUMBER       PICTURE X(4).
   02 FILLER                PICTURE X.
   02 CREATION              PICTURE X.
      03 YEAR               PICTURE 99.
      03 FILLER             PICTURE X.
      03 DAY                PICTURE 999.
   02 FILLER                PICTURE XXX.
   02 RETENTION-DAYS        PICTURE 999.
   02 FILLER                PICTURE X.
   02 FILE-DENSITY          PICTURE 9.
   02 FILE-MODE             PICTURE 9.
   02 CHK-SUM               PICTURE 9.
   02 SEQ-CHECK             PICTURE 9.
   02 CHECK-POINT           PICTURE 9.
   02 BCD-FILE-NAME         PICTURE X(18).
   02 SORT-RESERVED         PICTURE X(12).
   02 ARBITRARY             PICTURE X(12).
01 CUSTOMERS-NAME PICTURE X(12) VALUE
     'ABC WIDGETS'.
01 ID-WORD.
   02 TYPER PICTURE X VALUE ZERO.
      88 CK-RED-SEQ         VALUE 'Y'.
      88 CK-AND-RED         VALUE ' '.
      88 SEQ-AND-RED        VALUE 'Q'.
      88 RED-ONLY           VALUE '-'.
      88 CK-AND-SEQ         VALUE 'H'.
      88 CK-ONLY            VALUE '+'.
      88 SEQ-ONLY           VALUE '8'.
      88 NO-ERROR           VALUE '0'.
   02 FILLER PICTURE X(5) VALUE ZERO.
PROCEDURE DIVISION.
DECLARATIVES.
PLACE-CUST-NAME-IN-LBL SECTION.
   USE BEFORE STANDARD BEGINNING REEL
     LABEL PROCEDURE ON A-FILE.
```

```
           B. ENTER LINKAGE-MODE.
              CALL 'GETLBL' USING STANDARD-LABEL,
                   SIZE-OF-LABEL-IN-WORDS.
           C. ENTER COBOL.
           D. MOVE CUSTOMERS-NAME TO ARBITRARY
              OF STANDARD LABEL.
           E. ENTER LINKAGE-MODE.
              CALL 'PUTLBL' USING STANDARD-LABEL,
                   SIZE-OF-LABEL-IN-WORDS.
           F. ENTER COBOL.
           ERROR-IDENTIFY SECTION.
              USE AFTER STANDARD ERROR PROCEDURE
                   ON A-FILE.
           E11. ENTER LINKAGE-MODE.
              CALL 'ERRSET' USING ID-WORD.
           E12. ENTER COBOL.
           END DECLARATIVES.
           MAIN PROGRAM SECTION.
              OPEN INPUT A-FILE.
              DISPLAY 'THE LABEL FOUND UPON OPEN-
                   WAS . . . '.
              DISPLAY STANDARD-LABEL.
           MP1. NOTE MAIN PROGRAM DELETED FOR
                SAKE OF CLARITY.
              CUSTOMERS-NAME IS DETERMINED IN
                   THIS SECTION.
           INPUT-A-FILE SECTION.
              READ A-FILE AT END GO TO CLOSE-UP.
           TEST-FOR-DETECTED-ERR SECTION.
              IF NO-ERROR GO TO STOP-TESTING.
              IF SEQ-ONLY OR CK-AND-SEQ OR
                 SEQ-AND-RED OR CK-RED-SEQ DISPLAY
                 'SEQUENCE CKECK'.
              IF CK-ONLY OR CK-AND-SEQ OR
                 CK-AND-RED OR CK-RED-SEQ DISPLAY
                 'CHECKSUM'.
              IF RED-ONLY OR SEQ-AND-RED OR
                 CK-AND-RED OR CK-RED-SEQ DISPLAY
                 'READ REDUNDANCY'.
              DISPLAY 'ERROR(S) ENCOUNTERED WHILE
                 READING A-FILE'.
           STOP-TESTING.
              GO TO MP1.
           CLOSE-UP. CLOSE A-FILE. STOP RUN.
$CBEND
```

Map Subprogram Error Codes

```
$IBMAP    MAPDCK   FULIST, NODECK
ERRSET    SAVE
          LGL      3
          LDQ      =0
          LGR      3
          STQ*     3, 4          SAVE ERROR CODE BITS
          RETURN   ERRSET
          END
```

Map Subprogram Label Area

```
$IBMAP    GP       20, LIST, REF, NODECK
          ENTRY    GETLBL
GETLBL    AXT      2,2
          TRA      *+2
          ENTRY    PUTLBL
PUTLBL    AXT      1,2
          CLA      3,4
          STA      X
          CLA      4,4
          STA      *+1
          LAC      **,1
          TXL      ZERO,1,0
          SXD      *+10,1
          SXA      *+10,4
          AXT      ,1
          AXT      2,4
          XEC      Y+2,2
          XEC      Z+2,2
          TXI      *+1,1,-1
          TXH      *+2,1,-14
          AXT      1,4
          TXL      *+2,1,-21
          TXH      *-6,1,**
          AXT      **,4
          TRA      5,4
ZERO      AXT      -21,1
          TRA      *-14
W         PZE      .LAREA,1
          PZE      .AREA1-14,1
X         PZE      ,1
Y         CLA*     W+2,4
          CLA*     X
Z         STO*     X
          STO*     W+2,4
          END
```

Arrays and Subscripts

It is often necessary to have an entire array of information available for a COBOL program. An array can be set up in any of the three sections in the DATA DIVISION. Each entry in the array can be described and given a value in the CONSTANT SECTION; or the array can be developed during the execution of the program and the entries described in the WORKING-STORAGE SECTION; or the array can be read into storage from tape and described in the FILE-SECTION.

The following is an example of an array defined in the WORKING-STORAGE SECTION. Population figures are given for ten cities in each state. The states are divided into five regions, each region containing ten states.

```
01  POPULATION-TABLE.
  02  REGION1.
    03  MAINE USAGE IS COMPUTATIONAL.
      04  AUGUSTA PICTURE IS 99999
          SYNCHRONIZED RIGHT VALUE IS 34527.
      04  BANGOR PICTURE IS 99999 SYNCHRONIZED
          RIGHT VALUE IS 21438.
        .
        .
  02  REGION2.
        .
        .
```

These 556 entries in the WORKING-STORAGE SECTION put the entire array in storage, but calculations require a separate statement for each city. For example, to determine which cities have populations over 50,000, statements like the following are needed for each calculation:

```
IF AUGUSTA GREATER THAN 5000...
IF BANGOR GREATER THAN 50000...
```

Redefining the array allows the population of each city to be referred to by subscripting. The rules for subscripting are given in "Appendix A." The array can be redefined with the following entries, which immediately follow the entries defining the array.

```
01  TABLE REDEFINES POPULATION-TABLE.
  02  REGION OCCURS 5 TIMES.
    03  STATE OCCURS 10 TIMES.
      04  POPULATION OCCURS 10 TIMES PICTURE
          IS 99999 USAGE IS COMPUTATIONAL
          SYNCHRONIZED RIGHT.
```

Figure 31 shows the form of the array set up by these entries.

After redefinition, the population of Augusta, the first city in the first state in the first region, can be referred to as POPULATION (1, 1, 1), and Bangor can be referred to as POPULATION (1, 1, 2). Reference can be made to any part of the array. For instance, the whole third region can be referred to as REGION (3), or the second state in the third region can be referred to as STATE (3, 2).

Assigning names to these subscripts in the WORKING-STORAGE SECTION and varying the values in the PROCEDURE DIVISION allows a more general reference. For instance the constants AREA, PART, and CITY can be assigned initial values and varied in the PROCEDURE DIVISION so that reference can be made to POPULATION (AREA, PART, CITY).

In summary, the following entries can be used in the

```
DATA DIVISION.
FILE SECTION.
        .
        .
WORKING-STORAGE SECTION.
  77  AREA PICTURE IS 9 USAGE IS COMPUTATIONAL
      SYNCHRONIZED RIGHT VALUE IS 1.
  77  PART PICTURE IS 99 USAGE IS COMPUTATIONAL
      SYNCHRONIZED RIGHT VALUE IS 1.
  77  CITY PICTURE IS 99 USAGE IS COMPUTATIONAL
      SYNCHRONIZED RIGHT VALUE IS 1.
        .
        .
01  POPULATION-TABLE.
  02  REGION1.
    03  MAINE USAGE IS COMPUTATIONAL.
        .
        .
01  TABLE REDEFINES POPULATION-TABLE.
  02  REGION OCCURS 5 TIMES.
```

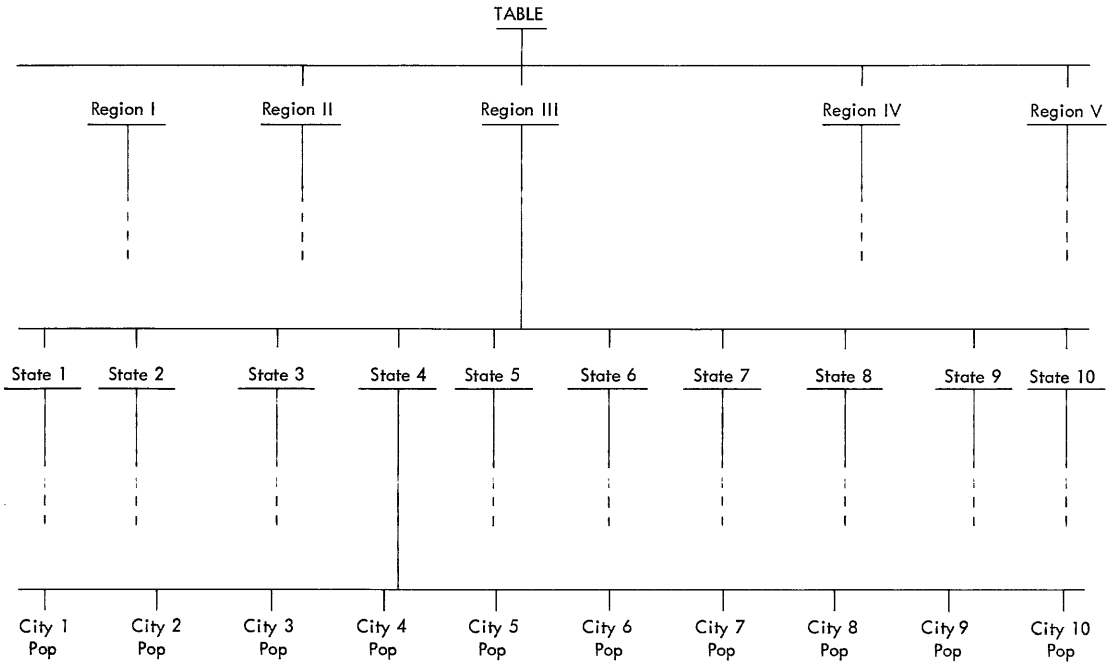


Figure 31. Organization of an Array Described with Nested OCCURS Clauses

```
03  STATE OCCURS 10 TIMES.
  04  POPULATION OCCURS 10 TIMES PICTURE
      IS 99999 USAGE IS COMPUTATIONAL
      SYNCHRONIZED RIGHT.
.
.
```

To determine which cities have populations over 50,000, the following statement can be used in the PROCEDURE DIVISION:

```
TEST. IF POPULATION (AREA, PART, CITY) IS
    GREATER THAN 50000 . . .
```

Option 5 of the PERFORM statement can be used to vary the subscripts AREA, PART, and CITY so that this one statement can be used for each city's population.

```
SUBSCRPT. PERFORM TEST VARYING AREA FROM 1
    BY 1 UNTIL
        AREA = 6 AFTER PART FROM 1 BY 1 UNTIL
        PART = 11 AFTER CITY FROM 1 BY 1 UNTIL
        CITY = 11.
```

The first time through the paragraph, POPULATION (AREA, PART, CITY) is equal to POPULATION (1, 1, 1); the eleventh time through the paragraph, it is equal to POPULATION (1, 2, 1), etc.

Blocking and Deblocking Records

If there are a large number of logical records in a block, it may be advantageous to block and deblock the records using the COBOL language. The basic technique is as follows:

1. Rather than describe the logical record after the file description entry, use the following entries.

ENTRIES	COMMENTS
01 data-name-1.	Data-name-1 is the name given in the DATA RECORDS clause, and *n* is the size of the block in words.
02 data-name-2 PICTURE X(6) OCCURS n.	

2. Describe the logical record in the WORKING-STORAGE SECTION and redefine it with a record of the following form:

ENTRIES	COMMENTS
01 logical-record.	
01 data-name-3 REDEFINES logical-record.	
02 data-name-4 PICTURE X(6) OCCURS m.	The letter *m* represents the size of the record in words.

3. To obtain or release records, give a PERFORM statement, rather than a READ or WRITE statement. The statements under control of the PERFORM are of the following type:

READ:

a. Using a counter as a subscript, MOVE subscripted the next *m* (record size) words from *data-name-2* to *data-name-4*.

b. When all *n* (block size) words have been moved, the statement READ file name AT END . . . causes a new block to be read.

WRITE:

a. Using a counter as a subscript, MOVE subscripted the next *m* (record size) words from *data-name-4* to *data-name-2*.

b. When all *n* (block size) words have been moved, the statement WRITE data-name-1 causes the block to be written out of storage.

4. If the last block on an output physical record is not completely filled with valid data, it may have to be padded before it is written out of storage. A test must be given to detect this padding for input records.

Example of Deblocking Records Using COBOL

The following example shows how variable-length records are deblocked. The block sizes are given in a dummy record description in the FILE SECTION. The logical record sizes are given in the WORKING-STORAGE SECTION. The general procedure follows the basic technique described for deblocking records. Separate MOVE statements (READ-INFILE SECTION and WRITE-OUT-RCD SECTION) are given instead of a PERFORM statement. This procedure takes less execution time but uses more storage.

The first record in the file is handled separately. Since the records are of variable length, a termination character appears at the end of each physical record. This is necessary only on the last record for fixed-length records.

Program Listing of Blocking and Deblocking Records:

```
$EXECUTE                IBJOB
$IBJOB                  LOGIC, MAP, FILES, LABELS
$IBCBC      COBOP       FULIST, DECK
            IDENTIFICATION DIVISION.
            PROGRAM-ID. TRACING PHASE OF
                CORPORATE EXPRESS PROGRAM.
            ENVIRONMENT DIVISION.
            CONFIGURATION SECTION.
            SOURCE-COMPUTER. IBM-7090.
            OBJECT-COMPUTER. IBM-7090.
            SPECIAL-NAMES.
               KEY 1 IS ABLEKEY ON STATUS IS SWISON.
            INPUT-OUTPUT SECTION.
            FILE-CONTROL. SELECT OUT-FILE ASSIGN TO
                A(1) FOR MULTIPLE REEL.
               SELECT INFILE ASSIGN TO B(1) FOR
                   MULTIPLE REEL.
               SELECT NEW-FILE ASSIGN TO B(1).
            DATA DIVISION.
            FILE SECTION.
            FD NEW-FILE RECORDING MODE IS BCD HIGH
                DENSITY
               BLOCK CONTAINS 1002 CHARACTERS
               LABEL RECORDS ARE OMITTED
               DATA RECORD IS DUM-REC.
            01 DUM-REC.
               02 RECS PICTURE X(18) OCCURS 55 TIMES.
               02 RECS-FILLER PICTURE X(12).
```

```
FD OUT-FILE
   RECORDING MODE IS BCD HIGH DENSITY
   BLOCK CONTAINS 1002 CHARACTERS LABEL
         RECORDS ARE OMITTED DATA
         RECORD IS OUT-RCD.
01 OUT-RCD.
02 OUT-WDS PICTURE IS X(06) OCCURS 167
         TIMES.
FD INFILE
   RECORDING MODE IS BCD HIGH DENSITY
   BLOCK CONTAINS 1002 CHARACTERS
   LABEL RECORDS ARE OMITTED
   DATA RECORD IS INRECORD.
01 INRECORD.
   02 IN-WDS PICTURE X(6) OCCURS 167
         TIMES.
WORKING-STORAGE SECTION.
01 WORKER.
   02 WORK-AREA PICTURE X(6) OCCURS 15
         TIMES.
01 WORKER-C REDEFINES WORKER.
   02 A1 PICTURE X(6).
   02 A2 PICTURE X(6).
   02 A3 PICTURE X(6).
   02 A4 PICTURE X(6).
   02 A5 PICTURE X(6).
   02 A6 PICTURE X(6).
   02 A7 PICTURE X(6).
   02 A8 PICTURE X(6).
   02 A9 PICTURE X(6).
   02 B1 PICTURE X(6).
   02 B2 PICTURE X(6).
   02 B3 PICTURE X(6).
   02 B4 PICTURE X(6).
   02 B5 PICTURE X(6).
   02 B6 PICTURE X(6).
01 WORKER-1 REDEFINES WORKER.
   02 PARTIAL PICTURE IS X(12).
   02 SOMEORE PICTURE IS X(12).
01 WORKER-2 REDEFINES WORKER.
   02 THISONE PICTURE IS X(6).
   02 THATTONE PICTURE IS X(12).
01 DUMMY.
   02 DUM-COUNT PICTURE 9(2)
         COMPUTATIONAL SYNCHRONIZED
         RIGHT VALUE IS 03.
   02 DUM-BAL PICTURE X(12) VALUE
         'ABCDEFGHIJKL'.
01 U PICTURE 9(5) SYNCHRONIZED RIGHT
      COMPUTATIONAL.
01 W PICTURE 9(5) SYNCHRONIZED RIGHT
      COMPUTATIONAL.
01 X PICTURE 9(5) SYNCHRONIZED RIGHT
      COMPUTATIONAL.
01 Y PICTURE 9(5) SYNCHRONIZED RIGHT
      COMPUTATIONAL.
01 Z PICTURE 9(5) SYNCHRONIZED RIGHT
      COMPUTATIONAL.
01 CLASSIFIER.
   02 KOUNT PICTURE 9(2) COMPUTATIONAL
         SYNCHRONIZED RIGHT.
PROCEDURE DIVISION.
STARTT.
   OPEN OUTPUT NEW-FILE.
   PERFORM CREATE-TAPE VARYING Y FROM
         1 BY 1 UNTIL Y = 601.
   CLOSE NEW-FILE.
   OPEN INPUT INFILE OUTPUT OUT-FILE.
   MOVE 1 TO W.
   MOVE 1 TO U PERFORM READ-NEW.
RIF.
   PERFORM READ-INFILE.
   PERFORM WRITE-OUT-RCD.
   GO TO RIF.
EOJ.
   CLOSE INFILE OUT-FILE.
   STOP 'KEY 1 DOWN TO RESTART'.
   IF SWISON GO TO STARTT.
   MOVE WORKER-2 TO DUMMY.
   DISPLAY DUMMY.
   STOP RUN.
READ-INFILE SECTION.
RI1.
   MOVE IN-WDS (X) TO CLASSIFIER.
   IF CLASSIFIER = SPACES PERFORM
         READ-NEW GO TO RI1.
   COMPUTE Y = X + KOUNT.
   MOVE IN-WDS (X) TO A1 ADD 1 TO X.
   IF X = Y GO TO END-1.
   MOVE IN-WDS (X) TO A2 ADD 1 TO X.
   IF X = Y GO TO END-1.
   MOVE IN-WDS (X) TO A3 ADD 1 TO X.
   IF X = Y GO TO END-1.
   MOVE IN-WDS (X) TO A4 ADD 1 TO X.
   IF X = Y GO TO END-1.
   MOVE IN-WDS (X) TO A5 ADD 1 TO X.
   IF X = Y GO TO END-1.
   MOVE IN-WDS (X) TO A6 ADD 1 TO X.
   IF X = Y GO TO END-1.
   MOVE IN-WDS (X) TO A7 ADD 1 TO X.
   IF X = Y GO TO END-1.
   MOVE IN-WDS (X) TO A8 ADD 1 TO X.
   IF X = Y GO TO END-1.
   MOVE IN-WDS (X) TO A9 ADD 1 TO X.
   IF X = Y GO TO END-1.
   MOVE IN-WDS (X) TO B1 ADD 1 TO X.
   IF X = Y GO TO END-1.
   MOVE IN-WDS (X) TO B2 ADD 1 TO X.
```

```
    IF X = Y GO TO END-1.
    MOVE IN-WDS (X) TO B3 ADD 1 TO X.
    IF X = Y GO TO END-1.
    MOVE IN-WDS (X) TO B4 ADD 1 TO X.
    IF X = Y GO TO END-1.
    MOVE IN-WDS (X) TO B5 ADD 1 TO X.
    IF X = Y GO TO END-1.
    MOVE IN-WDS (X) TO B6 ADD 1 TO X.
END-1. EXIT.
READ-NEW SECTION.
    READ INFILE AT END GO TO EOJ.
    MOVE 1 TO X.
WRITE-OUT-RCD SECTION.
    IF U + KOUNT GREATER THAN 167 MOVE
        SPACE TO OUT-WDS (U) WRITE
        OUT-RCD MOVE 1 TO U MOVE 1
        TO W.
COMPUTE W = W + KOUNT.
    MOVE 1 TO Z.
    MOVE A1 TO OUT-WDS (U) ADD 1 TO U.
    IF U = W GO TO END-2.
    MOVE A2 TO OUT-WDS (U) ADD 1 TO U.
    IF U = W GO TO END-2.
    MOVE A3 TO OUT-WDS (U) ALL 1 TO U.
    IF U = W GO TO END-2.
    MOVE A4 TO OUT-WDS (U) ADD 1 TO U.
    IF U = W GO TO END-2.
    MOVE A5 TO OUT-WDS (U) ADD 1 TO U.
    IF U = W GO TO END-2.
    MOVE A6 TO OUT-WDS (U) ADD 1 TO U.
    IF U = W GO TO END-2.
    MOVE A7 TO OUT-WDS (U) ADD 1 TO U.
    IF U = W GO TO END-2.
    MOVE A8 TO OUT-WDS (U) ADD 1 TO U.
    IF U = W GO TO END-2.
    MOVE A9 TO OUT-WDS (U) ADD 1 TO U.
    IF U = W GO TO END-2.
    MOVE B1 TO OUT-WDS (U) ADD 1 TO U.
    IF U = W GO TO END-2.
    MOVE B2 TO OUT-WDS (U) ADD 1 TO U.
    IF U = W GO TO END-2.
    MOVE B3 TO OUT-WDS (U) ADD 1 TO U.
    IF U = W GO TO END-2.
    MOVE B4 TO OUT-WDS (U) ADD 1 TO U.
    IF U = W GO TO END-2.
    MOVE B5 TO OUT-WDS (U) ADD 1 TO U.
    IF U = W GO TO END-2.
    MOVE B6 TO OUT-WDS (U) ADD 1 TO U.
END-2. EXIT.
CREATE-TAPE SECTION.
    PERFORM CREATE-IN VARYING Z FROM 1
      BY 1 UNTIL Z = 56.
    MOVE SPACES TO RECS-FILLER.
    WRITE DUM-REC.
CREATE-IN SECTION.
    MOVE DUMMY TO RECS (Z).
```

Partial List of Compiler Limitations

Size

1. There are 32,767 computer words in core storage in the 7090 and 7094 computers. Therefore, the following restrictions apply to the size of data:
   a. The record size is limited to 32,767 computer characters.
   b. The block size is limited to 32,767 computer words.
   c. Subscripts may not have values greater than 32,767.
   d. A data-item that controls the size of variable-length repeated items cannot have a value that is large enough to permit the repeated items to occupy more than 32,767 computer words. For example, if a data-item that occupies a *full* computer word (6 nonnumeric characters or 9 numeric computational digits) is described as OCCURS *integer* TIMES DEPENDING ON *data-name,* neither *integer* nor *data-name* may have a value greater than 32,767.

NOTE: Actually not all 32,767 words are available for data since the program is in storage too, as are various components of the operating system. The compiler does not issue a message unless these limits are exceeded, but if there is not enough core storage available, an error message is given when the program is loaded.

2. The compiler usually limits the size of elementary data-items to 2,048 computer characters. A data-item that occupies an integral number of computer words (e.g., a BCD item, synchronized, with the number of its characters evenly divisible by 6) may occupy a maximum of 2,048 computer words (12,288 characters). The language imposes further restrictions on the size of elementary items.

3. No more than 49 levels of qualifications are permitted.

Names

1. Each name in a COBOL program is entered in a compiler dictionary. Although the number of names permitted in the dictionary varies with the program, the maximum number ever permissible is 3,875.

The dictionary occupies 26,247 computer words in core storage. Not all of the words are available to the program since the compiler also makes entries in the dictionary. For example, each COBOL word used in the program is entered in the dictionary.

Every program-name uses from two to four dictionary words. Names with five or less characters occupy two words (plus a possibly shared pointer word). File-names and names that use the maximum of 30 characters occupy four words. Thus it is possible that the limit of the dictionary may be reached before 3,875 names are entered in it.

2. The *'entry-name'* used for cross reference with the ENTER verb is limited to six characters for compatability with other IBJOB subsystems.

3. The *'entry-name'* used in the ENTRY POINT form of the ENTER statement cannot be the same as the deck-name of the program that contains this form of the ENTER statement.

4. The maximum number of operands that can ever follow USING in an ENTER statement or that can follow a DISPLAY statement is 128. Actually, the practical limit varies from 50 to 128.

Files

1. The maximum number of files allowed in a single compilation is 63.

2. Nothing may be written on an input file. Checkpoints are written on labeled output files or on the system checkpoint unit.

3. Files attached to the card reader or card punch must be BCD recording mode, unblocked, not labeled, and of a fixed-length not exceeding 72 characters.

4. If the system units SYSIN1, SYSOU1, and SYSPP1 are attached as card units, the files must be described in accordance with the rules for card units.

Formulas and Conditional Expressions

1. A maximum of 500 arithmetic operands and operators is permitted in a single expression.

2. A maximum of 20 Boolean (logical) operators is permitted in a single conditional expression. Boolean operators are AND, OR, and NOT.

3. A maximum of 18 simple condition tests is permitted in a single conditional expression.

Miscellaneous

1. On the $IBCBC control card only three or seven index registers may be specified. Usually the 7090 has only three index registers.

2. Read redundancies on the system input tape dur-

ing compilation are retried 20 times before a disaster message is given.

Checklist

Order

1. The complete organization of a source program is given in "Appendix A."
2. A file description entry must precede the data-item descriptions of the data on the file.
3. Independent working-storage items should precede grouped working-storage items.
4. Independent constants should precede grouped constants.
5. A data-item description must begin with a level-number and data-name. A REDEFINES clause, if used, must follow the data-name. The other clauses may be written in any order.
6. The declarative section must be at the beginning of the PROCEDURE DIVISION.

Efficiency

1. The most efficient form for computational items is SYNCHRONIZED RIGHT so that the items do not have to be unpacked before they are used.
2. Subscripted references take more time to execute than direct references.
3. It is usually more efficient to use GO TO DEPENDING ON statements than a series of IF statements. If a data-name can have many values, assign code numbers rather than condition-names, since condition-names must be tested individually.
4. The use of six or fewer characters in names decreases compilation time and lessens the possibility of a name-table overflow.
5. It is usually more efficient to use COMPUTE statements than ADD, SUBTRACT, MULTIPLY, and DIVIDE, the individual arithmetic statements.
6. High blocking factors may be efficiently handled by describing the block as an unlabeled record in the file description entry and unblocking it with statements in the PROCEDURE DIVISION.
7. If a series of statements is to be executed repeatedly at only one point in a program, it is more efficient to place the statements where they are to be executed than to refer to them with a PERFORM statement. This is true only if the VARYING option of the PERFORM statement is not used.
8. If a data-item is to be filled with a single repeated character, it is more efficient to use a figurative constant than a literal. For example. MOVE ZEROES TO A is more efficient than MOVE 0 TO A if A is a scaled numeric item.
9. DISPLAY statements on the on-line printer are time consuming.
10. The computation of checksums (APPLY clause of the I-O-CONTROL paragraph of the ENVIRONMENT DIVISION) may considerably increase the running time of the program. The formation and checking of the block sequence number is less time consuming, however, and is helpful in detecting machine malfunctions.

Redundant Clauses

1. If CLASS is ALPHANUMERIC, USAGE is assumed to be DISPLAY.
2. If USAGE is COMPUTATIONAL, CLASS is assumed to be NUMERIC.
3. Edited items (described with a BLANK WHEN ZERO clause or by the report form of the PICTURE clause) are assumed to be ALPHANUMERIC DISPLAY.
4. An item containing a SIGNED clause is assumed to be NUMERIC.
5. It is redundant to specify an operational sign for an item whose USAGE is COMPUTATIONAL, either with a SIGNED clause or with an s in a PICTURE clause.
6. A PICTURE clause should not be used with POINT, SIGNED, or SIZE clauses. In case of a contradictory occurrence, an E level error message is produced, but the PICTURE specifications appear in the binary deck.
7. Computational floating-point numbers are not affected by a SYNCHRONIZED clause since they always occupy one (single-precision) or two (double-precision) full words.
8. The only clause required for computational floating-point items is USAGE IS COMPUTATIONAL-1 (or COMPUTATIONAL-2).
9. Any information provided in the CONSTANT SECTION can be put in the WORKING-STORAGE SECTION.

Input-Output

1. The BCD recording mode must be specified for a file assigned exclusively to card equipment.
2. The BCD recording mode must be specified for an output file assigned to a system unit (SYSOU1 or SYSPP1).
3. If BCD recording mode is specified for a file, the data on the file must be described (explicitly or implicitly) as USAGE DISPLAY.
4. If binary recording mode is specified for a file, the data on the file may be described as USAGE COMPUTATIONAL, COMPUTATIONAL-1, COMPUTATIONAL-2, or DISPLAY.
5. Two files that share the same file description entry; that is, the RENAMING clause in the FILE-CONTROL paragraph has been used, cannot be in OPEN status at the same time.
6. OPEN, CLOSE, and READ file-name. WRITE record-name.
7. Control words are required if records of different types are blocked on a file.
8. If records of a different type are on a file, the programmer must determine which record type has been read.

9. Checkpoints, not written on the checkpoint unit, are written on *labeled* output files only.

10. Logical records may not extend across physical records on a file.

11. Files assigned to card equipment may not be blocked.

12. If a file is to be processed for both input and output, there must be two SELECT entries for it in the FILE-CONTROL paragraph of the ENVIRONMENT DIVISION.

13. No input-output verbs may be used in the declaratives section of the PROCEDURE DIVISION.

14. If there is an error in a label, a message is written on-line. The machine operator has the option of ignoring the label error.

15. Tapes assigned to SYSOU1 and tapes to be printed should have a carriage control character preceding each record. For further information, see "Off-Line Processing of Output" in the publication *IBM 7090/7094 IBSYS Operating System, Operator's Guide*, Form C28-6355.

16. Depending on the way it is used most of the time, data should be described as COMPUTATIONAL or DISPLAY.

17. Blocks (BLOCK CONTAINS clause) must be longer than two computer words (12 computer characters), or they cannot be read or written.

18. If blocks (BLOCK CONTAINS clause) are not composed of an integral number of computer words, the rightmost portion of a record on an output file may contain unwanted information since the full word is to be written.

Level-Numbers

1. The 01 level-number must be used for the highest levels of data organization in the data-item descriptions in the FILE and WORKING-STORAGE SECTIONS.

2. The OCCURS clause may not be used with an entry with a 01 or 77 level-number.

3. A condition-name, identified by the level-number 88, must have a VALUE clause.

4. SIGNED, SYNCHRONIZED, POINT, PICTURE, and BLANK WHEN ZERO may be specified only at an elementary level.

5. A level-number is required at the beginning of every data-item description.

Size

1. A maximum of 2,048 characters is usually permitted for alphanumeric elementary data-items. If the item occupies an integral number of computer words, the maximum size is 12,288 characters.

2. The maximum size for computational (not floating-point) numbers is 18.

3. The value of the mantissa of single-precision floating-point numbers may not exceed eight digits. The exponent may range in value from −38 to +38.

4. The contents of the mantissa of a double-precision floating-point number may not exceed 16 digits. The exponent many range in value from −38 to +38.

5. The maximum size of numeric literals is 18; the maximum size of nonnumeric literals is 120.

6. The maximum number of character codes in a PICTURE is 30.

7. Operational signs, assumed decimal points, and assumed character positions (specified by S, V, and P, respectively, in the PICTURE clause) are not counted in determining the size of an item.

8. When the CHARACTERS form of the BLOCK CONTAINS clause is used, control words indicating the length of the records must be counted in determining the size of the block. If the APPLY clause of the I-O-CONTROL paragraph has not been used, and if there are block sequence words on the file, block sequence words must be counted in determining the size of the block.

Data-Values

1. Subscripts must have positive integral values greater than zero.

2. Exponentiation of a negative value is allowed only if the exponent is a nonnegative integer. Otherwise, the result of the exponentiation is zero.

3. A VALUE clause may not be used with a:
   a. REDEFINES clause except for condition-name entries.
   b. data-item described after an entry in the same logical record that contains an OCCURS DEPENDING ON clause.
   c. data-item description that contains an OCCURS clause.
   4. data-item that is a subdivision of an entry containing an OCCURS clause.

4. In the FILE SECTION, the VALUE clause may be used with condition-names or to provide label information. In the CONSTANT SECTION, the VALUE clause may be used only to specify the initial value of storage. In the WORKING-STORAGE SECTION, the VALUE clause may be used with condition-names or to specify the initial value of storage.

General

1. Computational floating-point numbers must be described as COMPUTATIONAL-1 or COMPUTATIONAL-2 in the USAGE clause. The scientific decimal form of the PICTURE clause is used only for editing.

2. Floating-point literals may be used only in the DATA DIVISION.

3. A REDEFINES clause must not be used with logical records on the same file. A DATA RECORDS clause in the file description entry provides automatic redefinition.

4. When a group item is manipulated in the PROCEDURE DIVISION, it is treated as an elementary item with a PICTURE of all X's.

5. The statements following the AT END clause of the READ verb and the ON SIZE ERROR clause of the arithmetic verbs must be imperative, not conditional.

6. An ON SIZE ERROR clause should be included in arithmetic statements whenever there is a possibility that a size error might occur.

7. If zero is divided by zero, the result is zero.

The COBOL Compiler is a component of the IBJOB Processor, which is, in turn, a component of the IBSYS Operating System. Figure 32 shows the logical structure of the IBSYS Operating System.

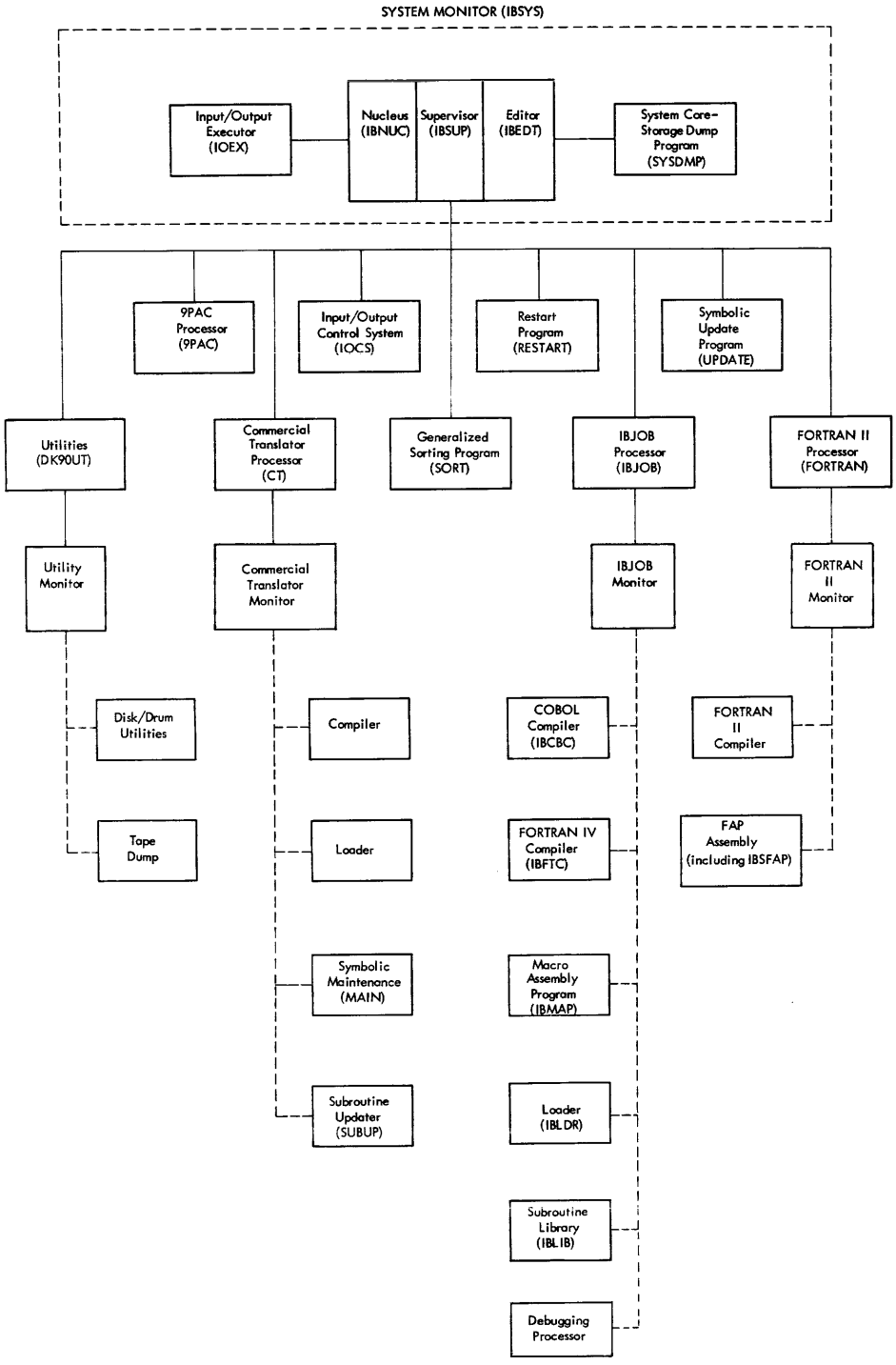


Figure 32. The IBSYS Operating System

Monitored operation in the IBSYS Operating System is directed by control cards. The IBSYS *System Monitor* controls communication among the components of the IBSYS Operating System. The IBJOB *Processor Monitor* controls communication between the System Monitor and components of the IBJOB Processor.

The publication *IBM 7090/7094 IBSYS Operating System: IBJOB Processor*, Form C28-6389, contains a description of all IBJOB Processor control cards.

Control Cards Used in a COBOL Program

The following control cards may be used to compile and execute a COBOL program. Underlined options in the formats of the control cards are assumed, if the option is omitted.

Figures 33 and 34 show the order of control cards used for compiling and executing a COBOL program.

$JOB Card

The IBJOB Processor Monitor transfers control to the IBSYS System Monitor when the Processor Monitor reads a $JOB card. If system units have not been reassigned or made unavailable during the last job and if a between-jobs interrupt condition does not exist, the Processor Monitor regains control and transfers to the installation accounting routine. If there is no installation accounting routine, the $JOB card is listed on both the system printer and the system output unit and the System Monitor retains control.

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

The format of the $JOB card is:

1	16
$JOB	any text

Columns 16 through 72 are normally used to identify a job and may contain any combination of alphameric characters and blanks.

$EXECUTE Card

The $EXECUTE card must precede a COBOL program within a job if one of the following conditions exists:

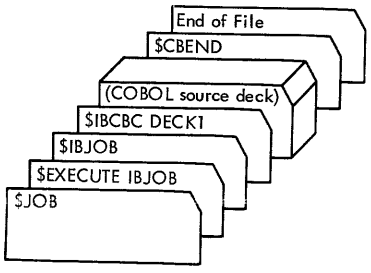
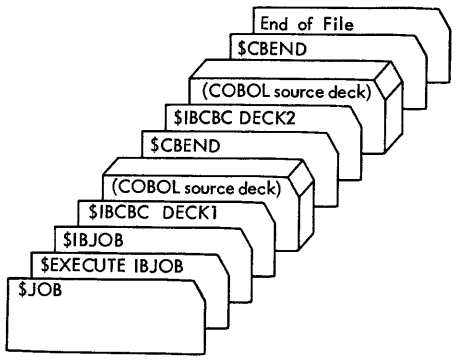


Figure 33. Sample Control Card Deck for One COBOL Compilation



NOTE: Any number of COBOL source decks or MAP or FORTRAN IV source decks may appear between the $IBJOB and the End-of-File card.

Figure 34. Sample Control Card Deck for Two COBOL Compilations

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

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

The format of the $EXECUTE card is:

1	16
$EXECUTE	subsystem name

The subsystem name may consist of six or fewer BCD characters beginning in column 16. For COBOL programs, the subsystem name is IBJOB.

$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 the $IBJOB control card prescribe the manner in which an application may be processed.

The format of the $IBJOB card is:

1	16
$IBJOB	[, options]

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

Execution Option

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

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

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

Logic Options

[, { NOLOGIC / LOGIC / DLOGIC }]

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

2. LOGIC — A cross-reference table of the program sections and of the system subroutines required for execution is generated. The origin and length of each program section and subroutine, and the buffer assignments are also given. When this option is specified for a Librarian execution, a listing of the control section dependencies in the generated library is produced.

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

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

MAP Options

[, { NOMAP / MAP }]

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

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

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

File List Options

[, { NOFILES / FILES }]

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

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

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

Input Deck Options

[, { SOURCE / NOSOURCE }]

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 and/or assembly is required in the application (SOURCE).

Input/Output Options

[, { IOEX / MINIMUM / BASIC / LABELS / FIOCS / ALTIO }]

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

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 loaded with the object program. When this option is specified, the IOCS package intended especially for use with FORTRAN IV programs is loaded with the standard package.

If none of these options is specified, the Loader determines the level of IOCS to be loaded with the object program.

5. FIOCS — The standard FORTRAN IV input-output package is loaded with the object program.

6. ALTIO — The alternate FORTRAN IV input-output package is loaded with the object program.

The levels of IOCS are described in the publication *IBM 7090/7094 IBSYS Operating System: Input/Output Control System*, Form C28-6345.

Overlay Options

[, { FLOW / NOFLOW }]

1. FLOW — Execution of the object program is not permitted if the rules concerning references between links are violated. These rules may be found in the section "Overlay Feature of the Loader" in the publication *IBM 7090/7094 IBSYS Operating System: IBJOB Processor*, Form C28-6389.

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 (FLOW).

$IBCBC Card

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

The format of the $IBCBC card is:

1	8	16
$IBCBC	deckname	[, options]

where *deckname* names the deck that follows. A deck name of six or fewer 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 embedded blanks.

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

List Options

[, { NOLIST / LIST / FULIST }]

1. NOLIST – A listing of the object program is not wanted. Error messages produced by the Assembler are listed.

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

3. FULIST – A listing of the object program is generated, one instruction per line. This listing includes generated octal information. The following is an example of a program listing:

```
00041  0604 00 0 01400  10011  STI   .CAREF
00042  0441 00 0 00150  10001  LDI   SP+1
00043  0604 00 0 02000  10011  STI   .CBREF
00044  0074 00 4 02400  10011  TSK   .CMPK3,4
00045  1000006 1 03000  10011  TXI   .CANA2,1,6
00046  1000004 1 03400  10011  TXI   .CANA3,1,4
00047  0441 00 0 00154  10001  LDI   SP+5
00050  0604 00 0 01400  10011  STI   .CAREF
```

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

Symbol Table Options

[, { NOREF / REF }]

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

2. REF – A sorted dictionary of the source language names and their associated Equivalent Name (EN) numbers and a cross-reference table of the symbols used in the object program are generated. The following is an example of a cross-reference dictionary:

```
FIELD1                EN0235
FIELD2                EN0237
OUTPUT-FILE           EN0224, 0230
OUTPUT-RECORD         EN0231
WORK-RECORD           EN0234
```

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

Punch Options

[, { DECK / NODECK }]

1. DECK – The object program is written on the System Peripheral Punch Unit (SYSPP1).

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 (DECK).

Instruction Set Options

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

1. M90 – The object program uses only 7090 machine instructions. All double-precision operations are simulated by system macros, and EVEN pseudo-operations are treated as commentary.

2. M94 – The object program uses only 7090 machine instructions.

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

NOTE: Only the M90 option has any effect. The other options have not been implemented. If 7 index registers are to be used, XR7 must be specified.

Index Register Options

[, { XR3 / XR7 }]

1. XR3 – The object program uses three index registers (1, 2, and 4).

2. XR7 – The object program uses 7 index registers.

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

Code Options

[, { INLINE / TIGHT }]

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

2. TIGHT – The object program's computational and MOVE tasks that are generated are smaller, thereby conserving object-time core storage.

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

Tape Error Options

[, { IOEND / READON }]

1. IOEND – Errors that occur while reading tape at object time cause irrecoverable error conditions.

2. READON – Errors that occur while reading tape at object time are ignored. This option allows high-volume data processing to continue while ignoring low-volume error conditions.

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

Collating Sequence Options

[, {COMSEQ / BINSEQ}]

1. COMSEQ – The object program uses the commercial collating sequence.

2. BINSEQ – The object program uses the binary scientific collating sequences.

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

Debugging Dictionary Options

[, {NODD / DD}]

1. NODD – A debugging dictionary is not wanted.

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

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

$CBEND Card

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

The format of the $CBEND card is:

1	8
$CBEND	

Debugging Package

The debugging package enables the programmer to manipulate data, control processing, and obtain dumps of the contents of any locations in core storage at specified locations within his program. There is no limit on the number of requests that may be given for a single program.

This package provides the programmer with two types of debugging: compile-time debugging and load-time debugging. The publication *IBM 7090/7094 IBSYS Operating System: IBJOB Processor Debugging Package*, Form C28-6393, contains examples of debugging requests.

Compile-Time Debugging

Compile-time debugging may be used to analyze and display various results in a COBOL source program. Debug requests are similar to COBOL procedural statements and almost all procedural capabilities of the compiler may be utilized.

$IBDBC Card

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

The format of the $IBDBC card is:

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

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

Location is the COBOL section-name or paragraph-name (qualified, if necessary) indicating the point in the program at which the request is to be executed (insertion point). Effectively, debug requests are performed as if they were physically placed in the source program following the section- or paragraph-name, but preceding the text associated with the name.

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

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

An end-of-file card or any $-control card terminates the compile-time debugging packet. The debugging packet follows the $CBEND control card in a COBOL source program.

Count-Conditional Statement

The count-conditional statement, which can be used only in debug requests, provide a means of determining when debugging action should be taken. For example, it can be used to cause a debug statement to be executed the first, third, fifth, etc., times through a loop that begins at a paragraph-name specified by *location* on the $IBDBC control card. The count-conditional statement has the same structure as the IF statement and may be nested in the same manner. The format of the count conditional statement is:

ON positive-integer-1 [AND EVERY positive-integer-2] [UNTIL positive-integer-3] statement-1 [{ELSE / OTHERWISE} statement-2]

Positive-integer-1 specifies the time that *statement-1* is to be executed for the first time. For example, in the following debug request:

```
$IBDBC                PARAG1
    ON 3 DISPLAY A.
```

cause the statement DISPLAY A to be executed on the third time through PARAG1 (probably the paragraph-name of a series of statements executed under control of a PERFORM verb). No action is taken at any other time.

Positive-integer-2 and *positive-integer-3* specify the intermediate times and the last time *statement-1* is to be executed. For example, the following debug request:

```
$IBDBC                PARAG1
    ON 5 AND EVERY 3 UNTIL 12 DISPLAY A.
```

causes the statement DISPLAY A to be executed on the fifth, eighth, and eleventh time the statements at PARAG1 are executed. No action is taken at any other time.

Statement-2 is used to specify an action for the times *statement-1* is not executed. For example, the following debug request:

```
$IBDBC                PARAG1
    ON 2 AND EVERY 2 UNTIL 10 DISPLAY A ELSE
        DISPLAY B.
```

causes the statement DISPLAY A to be executed on the second, fourth, sixth, and eighth times through PARAG1 and the statement DISPLAY B to be executed at all other times.

Load Time Debugging

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

The following alphabetic list contains the error messages generated by the COBOL Compiler and their explanations. Messages involving the linkage-mode language and compile-time debugging language are included in this list, since they are so closely connected to COBOL programming.

In addition to the error message, a card number corresponding to the number of a card in the deck is also generated. This number does not necessarily mean that the error occurred on the particular card but only in that general area. If the compiler for some reason cannot determine the number of a card, it lists the number as either 0000 or 9999. External names of the form ENxxxx (Equivalent Names) are also generated and printed on the output listing. The numbers correspond to names assigned by the programmer in the DATA DIVISION of a COBOL program. They are frequently used for reference in the error messages.

Each error message is preceded by a code indicating the level of the error.

W	(Warning)	Compilation and execution continue.
E	(Error)	Execution is suppressed.
D	(Disaster)	Compilation is terminated.

Words in a message that must vary from situation to situation are denoted by "*****." Where asterisks actually appear as a standard part of a message, the condition is specifically noted.

A SPACE SHOULD SEPARATE A SUBSCRIPTED NAME FROM THE FOLLOWING LEFT PARENTHESIS. SPACE IS ASSUMED.
Self-explanatory.

'ACCEPT' MAY ONLY BE FOLLOWED BY A DATA-NAME. NOTHING DONE.
See the ACCEPT verb.

ALL CHARACTERS ACCEPTED FOR ***** MUST BE NUMERIC.
See the ACCEPT verb.

ALPHABETIC CLASS SPECIFIED FOR ***** IGNORED SINCE ITEM IS EXTERNAL DECIMAL.
ALPHABETIC CLASS cannot be specified for external decimal (NUMERIC DISPLAY) items.

ALPHABETIC OR ALPHANUMERIC CLASS SPECIFIED FOR ***** IGNORED SINCE ITEM IS INTERNAL DECIMAL.
Internal decimal (NUMERIC COMPUTATIONAL) items cannot be alphabetic or alphanumeric.

ALTER AT ***** DISALLOWED SINCE IT IS NOT SINGLE GO TO SENTENCE.
See the ALTER verb.

ALTER REFERENCE INCORRECT ***** IS NOT A ***** NOTHING DONE.
There has been incorrect use of the ALTER verb. The ALTER statement is ignored.

***** AND ***** HAVE NO CORRESPONDING SUBFIELDS. NO ACTION STATEMENTS GENERATED FOR THIS PAIR.
See the MOVE CORRESPONDING verb.

ARGUMENT NUMBER ***** MAY NOT APPEAR IN A DISPLAY STATEMENT. SPACE ASSUMED INSTEAD.
See the DISPLAY verb.

ARITHMETIC PHRASES IN CONDITIONAL EXPRESSIONS MAY NOT CONSIST OF MORE THAN 500 OPERATORS AND OPERANDS. EXPRESSION DELETED SINCE LIMIT EXCEEDED.
Break expression into smaller parts.

*****, ASSOCIATED WITH OCCURS ... DEPENDING ON ..., IS AN IMPROPER DATA ITEM. CLAUSE IGNORED.
The data-name is required to be a positive integer greater than zero. See OCCURS clause of the data-item description.

*****, ASSOCIATED WITH REDEFINES OR OCCURS ... DEPENDING ON ..., IS AN IMPROPER DATA ITEM. CLAUSE IGNORED.
See the REDEFINES and OCCURS clauses of the data-item description.

ATTEMPTED DIVISION BY ZERO BYPASSED. RESULT TAKEN TO BE ZERO.
Division by zero is mathematically undefined.

BINARY COMPUTATIONAL USAGE OF ***** INCOMPATIBLE WITH BCD RECORDING MODE FOR THIS FILE.
The record must agree with the mode specifications.

BINARY RECORDING MODE SPECIFICATION OF FILE ***** ASSIGNED TO CARD UNIT IS NOT PERMITTED.
Cards coming from the card reader must not be in binary.

BLOCK SIZE (***** COMPUTER WORDS) SPECIFIED FOR FILE ***** IS NOT A MULTIPLE OF RECORD SIZE (***** COMPUTER WORDS). BLOCK SIZE CHANGED TO ***** COMPUTER WORDS.
This message indicates that the block size and the record size are inconsistent. See the BLOCK CONTAINS and RECORD CONTAINS clauses.

BLOCKING OF DISTINCT RECORD TYPES OF DIFFERING SIZES, WITHOUT COUNT CONTROL, IN FILE ***** IS NOT PERMITTED. FILE IS SET UNBLOCKED.
See BLOCK clause of file description entry.

***** CANNOT BE SUBSCRIPTED. SCAN RESUMED AT NEXT VERB, PERIOD, OR INFORMATION IN THE A MARGIN.
See subscripts.

***** CANNOT BE USED AS AN ARGUMENT FOR THE CORRESPONDING OPTION.
See the CORRESPONDING option of the ADD, SUBTRACT, and MOVE verbs.

***** CANNOT HAVE MORE THAN 49 QUALIFIERS. EXTRA ONES DELETED.
Self-explanatory.

CANNOT USE VARIABLE LENGTH ITEMS FOR COMPARISON. NOTHING DONE.
See IF conditional statements in the PROCEDURE DIVISION.

CARD SEQUENCE ERROR IN COLUMNS 1-6. CONDITION IGNORED.
The card sequence has been checked and this error message results. There is no effect on compilation.

CARD UNIT NOT ALLOWED AS SECONDARY UNIT ASSIGNED TO FILE *****. SECONDARY UNIT ASSIGNMENT IGNORED.
See the FILE-CONTROL paragraph in the INPUT-OUTPUT section of the ENVIRONMENT DIVISION.

CAUTION, GROUP ITEM ***** TESTED.
A group item was an operand of an EXAMINE or IF class-test-type statement. This is a warning message.

CAUTION, GROUP LEVEL MOVE FROM ***** TO *****.
See the MOVE CORRESPONDING verb.

CAUTION, MOVE FROM ***** TO ***** CAUSES TRUNCATION.
This message indicates that either the size or number of decimal places of the items did not match. There is a possibility that information will be lost.

CAUTION, MOVE FROM ***** TO ***** CAUSES TRUNCATION EXCEPT IN CASES OF SYNCHRONIZATION.
This message indicates that there might be a loss of significant data.

CHARACTER LOGIC MOVE INVOLVING AN ITEM LONGER THAN 32767 CHARACTERS. NOTHING GENERATED.
This message indicates a compiler limitation has been reached.

CHECKPOINTS DESIGNATED TO BE WRITTEN ON FILE ***** BUT FILE IS NOT LABELED OUTPUT. CHECKPOINTS WILL BE WRITTEN ON STANDARD CHECKPOINT UNIT INSTEAD.
This message indicates that checkpoints cannot be written on an input file or an unlabeled output file.

CLOSE REEL FOR ***** IS ILLEGAL SINCE FILE IS ASSIGNED TO A CARD OR SYSTEM UNIT. REEL OPTION IGNORED.
Self-explanatory.

COBOL COMPILER DOES NOT OBEY THE USE OF XR4, XR5, OR XR6 ON $IBCBC CARD. XR3 IS ASSUMED.
Index register 3 and 7 are the only index register specifications accepted. See control cards.

COBOL WORD ***** WAS NOT FOUND WHERE REQUIRED IN THIS STATEMENT. STATEMENT DELETED.
This message indicates a language violation. See the rules regarding the use of the particular verb used.

COBOL WORD 'SECTION' MISSING. BEGINNING OF ***** SECTION ASSUMED BY COMPILER.
See "Organization of Source Program" and the ENVIRONMENT DIVISION.

COBOL WORDS 'ASSIGN TO' OMITTED IN SELECT ENTRY ***** ASSUMED UNIT ASSIGNMENTS IS '1 TAPE-UNIT'.
See the FILE-CONTROL paragraph in the INPUT-OUTPUT SECTION of the ENVIRONMENT DIVISION.

COBOL WORDS 'TO PROCEED TO' NOT FOUND WHERE REQUIRED IN ALTER STATEMENT. STATEMENT DELETED.
See the ALTER verb.

'COLLATE-COMMERCIAL' SHOULD NOT APPEAR IN ENVIRONMENT DIVISION. COLLATING SEQUENCE ASSUMED COMMERCIAL UNLESS 'BINSEQ' APPEARS ON $IBCBC CARD.
This message indicates obsolete wording.

COMMA ILLEGALLY TO RIGHT OF POINT IN PICTURE OF REPORT ITEM *****. + + + + + IS ASSUMED PICTURE.
See the PICTURE clause of the data-item description.

COMPILER ***** COUNT CONTROL CONVENTION ***** FILE *****.
See the file description entry in the DATA DIVISION.

COMPILER ***** COUNT CONTROL CONVENTION ***** FILE ***** UNLESS ***** IS ASSIGNED TO A CARD UNIT AT OBJECT-TIME.
See the file description entry in the DATA DIVISION.

COMPILER ALLOWS ONLY 20 CONSECUTIVE IMPLIED BOOLEAN OPERATORS. CONDITIONAL EXPRESSION DELETED SINCE MAXIMUM EXCEEDED.
Break the expression into smaller parts.

COMPILER ASSUMES FILE (S) ASSOCIATED WITH FD ENTRY ***** HAS LABEL RECORD SINCE VALUE OF LABEL GIVEN.
See the LABEL and VALUE clauses of the file description entry.

COMPILER BASE LOCATOR CAPACITY EXCEEDED. TRY SUBDIVIDING INTO SMALLER PROGRAMS FOR SEPARATE COMPILATION WITH COMBINATION AT OBJECT TIME.
This message indicates a compiler limitation has been reached.

COMPILER FORCED TO ASSUME ***** IS A GROUP ITEM DUE TO ERROR IN SUBSEQUENT LEVEL NUMBER.
See level-numbers in the DATA DIVISION.

COMPILER IGNORES ILLEGAL CLAUSES IN DESCRIPTION OF LEVEL 88 CONDITION ***** (ONLY VALUE IS ALLOWED).
See condition-name in the DATA DIVISION.

COMPILER TABLE CAPACITY EXCEEDED. TRY SUBDIVIDING INTO SMALLER PROGRAMS FOR SEPARATE COMPILATION WITH COMBINATION AT OBJECT TIME.
This message indicates that a compiler limitation has been exceeded. Suggested action is included in the message.

COMPILER THWARTED IN SEARCHING DATA STRUCTURE FOR GROUP(S) CONTAINING ARRAY ***** PROBABLY DUE TO TOO MANY SUBSCRIPTS GIVEN. OBJECT PROGRAM USES FIRST ELEMENT OF THE ARRAY.
See subscripts.

CONDITIONAL EXPRESSION TEST CAPACITY EXCEEDED. REWRITE AS TWO OR MORE SEPARATE EXPRESSIONS WITH A MAXIMUM OF 18 OPERATORS. ILLEGAL SENTENCE STRUCTURE. NOTHING DONE.
This message indicates that a compiler limitation has been reached.

CONDITIONAL VARIABLE ***** IMPROPERLY DESCRIBED AS A REPORT, SCIENTIFIC DECIMAL, OR FLOATING POINT ITEM. X IS ASSUMED PICTURE.
See conditional variables.

CONFLICTING 'USE' OPTIONS FOR FILE ***** OVERRIDDEN BY 'USE' STATEMENT(S) FOR ***** FILES.
This message indicates that conflicting USE procedures have occurred. See the USE verb.

CONTINUATION CHARACTER MUST NOT BE USED WITH AN OCCUPIED A MARGIN. CONTINUATION CHARACTER IGNORED.
Continued items must not begin before B margin. See "Reference Format."

CONTROL CARD ENCOUNTERED PRECEDING $CBEND CARD. END OF COBOL TEXT ASSUMED.
It is assumed that the $CBEND card is missing.

COPY OPTION NOT IMPLEMENTED. FD ENTRY IGNORED.
Enter the information in detail.

CORRESPONDING FIELDS OF ***** AND ***** OVERLAP.
See the MOVE CORRESPONDING verb.

CROSS-REFERENCE NAME TOO LONG. FIRST 6 CHARACTERS USED.
A cross-reference name is limited to 6 characters. See the ENTER verb.

DATA DIVISION-HEADER NOT FOLLOWED BY SECTION-NAME. SCAN RESUMED AT NEXT DATA DESCRIPTION ENTRY, SECTION, OR DIVISION.
See the "Organization of Source Program" and DATA DIVISION.

DATA ITEM ***** INVALID AS AN ARGUMENT IN 'EXAMINE' STATEMENT OR CLASS TEST. STATEMENT DELETED.
See the EXAMINE verb.

DATA ITEM ***** IS UNDER INFLUENCE OF INCONSISTENT USAGE AND CLASS CLAUSES. DETERMINING HIERARCHY IS PICTURE, USAGE, CLASS.
See the data-item description in the DATA DIVISION.

DATA ITEM ***** WITH REDEFINES CLAUSE NOT PRECEDED BY AN ELEMENTARY ITEM. REDEFINES IGNORED.
The redefining item must follow the redefined item with no intervening entries. See the REDEFINES clause of the data-item description.

DATA-NAME, NOT *****, EXPECTED AS ARGUMENT IN THIS STATEMENT. STATEMENT DELETED.
See the rules regarding the use of the particular verb referred to in this message.

DATA-NAME ***** REQUIRING CONVERSION, EDITING, OR DEFINITION MAY NOT APPEAR IN AN ACCEPT STATEMENT. NOTHING DONE.
See the ACCEPT verb.

DATA RECORDS CLAUSE OMITTED IN FD ENTRY *****. CONDITION IGNORED.
See the DATA RECORDS clause of the file description entry.

DECK NAME IN 'CALL' STATEMENT MUST BE ENCLOSED IN QUOTES. STATEMENT DELETED.
This message indicates a language rule violation. See the ENTER verb.

DECK NAME IS MISSING ON $IBCBC CARD. CONDITION IGNORED.
Self-explanatory.

'DECLARATIVES' MUST BE AT BEGINNING OF PROCEDURE DIVISION. STATEMENT DELETED.
This message indicates a language requirement. See "Organization of Source Program."

***** DEFINED IN MORE THAN ONE OUTPUT FILE. INPUT/OUTPUT STATEMENT IGNORED.
The record name used must be unique.

***** DESCRIPTION ENTRY ENCOUNTERED. BEGINNING OF ***** SECTION ASSUMED BY COMPILER.
See "Organization of Source Program" and the DATA DIVISION.

DISCREPANCY BETWEEN LEVELS OF ***** AND THE REDEFINED ITEM. DISCREPANCY IGNORED AT THIS POINT OF ANALYSIS.
The redefining item should match the redefined item. In this case, the level numbers do not match. This is a warning level message. See the REDEFINES clause of the data-item description.

DIVISION NAME SHOULD BE FOLLOWED BY THE WORD DIVISION AND A PERIOD. CONDITION IGNORED.
Self-explanatory.

DIVISION MUST BE IN ORDER AND NOT DUPLICATED. COMPILER SKIPS TO NEXT DIVISION.
See "Organization of Source Program."

$ IS A LEGAL CHARACTER ONLY IN THE PICTURE CLAUSE. $ DELETED.
Self-explanatory.

DOUBLE ASTERISKS (INDICATING EXPONENTIATION) SHOULD NOT BE SEPARATED BY SPACE(S). SPACE(S) IGNORED.
See formulas in the COMPUTE statement in the PROCEDURE DIVISION.

DOWNSCALE GENERATED WHICH LOSES ALL SIGNIFICANT DIGITS.
This message indicates that data has been lost due to downscaling.

ELEMENTARY LEVEL CLAUSES IN DESCRIPTION OF GROUP ITEM ***** IGNORED (I.E., VALUE, SIGNED, POINT, SYNCHRONIZED, EDITING, OR PICTURE.)
Certain clauses apply only to elementary items. See the discussion of the particular clause.

END OF COBOL MESSAGES.
Self-explanatory.

ENVIRONMENT ***** NOT FOLLOWED BY ***** SCAN RESUMED AT NEXT PARAGRAPH, SECTION, OR DIVISION.
See "Organization of Source Program" and the ENVIRONMENT DIVISION.

ENVIRONMENT PARAGRAPH-NAME ENCOUNTERED. BEGINNING OF ***** SECTION ASSUMED BY COMPILER.
See the ENVIRONMENT DIVISION.

ERRONEOUS PARENTHESIZATION IGNORED. TRANSLATION CONTINUES ARBITRARILY.
The number of left parentheses should equal the number of right parentheses.

ERROR IN OCCURS ... DEPENDING ON CLAUSE IN DESCRIPTION OF *****. COMPILER ASSUMES OCCURS EXACTLY 100 TIMES, IGNORING QUANTITY ITEM.
See the OCCURS clause in the data-item description entry.

ERRORS IN OCCURS ... DEPENDING ON CLAUSE IN DESCRIPTION OF *****. COMPILER IGNORES 'INTEGER-1 TO' SPECIFICATION.
See the OCCURS clause.

ERROR MESSAGE NOT YET IN FILE.
This message should never appear. If it does, a systems engineer should be called to initiate an error report.

ERROR NUMBER ** DUMMY **
This message precedes a debugging printout.

EXCESSIVE BLOCK SIZE SPECIFIED FOR FILE *****. FILE IS SET UNBLOCKED.
See the BLOCK CONTAINS clause of the file description entry.

EXTRANEOUS 'ELSE' FOUND. IF IN 'AT END' OR 'ON SIZE ERROR' CLAUSE, 'ELSE' TREATED AS A PERIOD, IN OTHER CASES IT IS TREATED AS A COMMA.
This message indicates a language violation. See the rules regarding the particular verb for which ELSE is an option. The statement must be rewritten to eliminate the extraneous ELSE.

FD ENTRY ***** NOT TERMINATED BY A PERIOD. CONDITION IGNORED.
Self-explanatory.

FIGURATIVE CONSTANT OR NON-NUMERIC LITERAL MUST FOLLOW 'ALL.' ALL ZERO ASSUMED IF COBOL WORD NEXT.
See figurative constants.

FILE ***** ASSIGNED TO *****, BUT FILE IS NOT ***** ***** USAGE ASSUMED.
File usage is contradictory to system unit usage.

FILE ***** ASSIGNED TO CARD UNIT HAS RECORD CONTAINING MORE THAN 72 CHARACTERS. MAXIMUM RECORD SIZE PROCESSED IS 72 CHARACTERS.
Card column limitation has been exceeded. There are only 72 columns available. The first 72 characters will be processed.

FILE ***** ASSIGNED TO SYSOU1, BUT RECORDING MODE GIVEN AS BINARY. RECORDING MODE CHANGED TO BCD.
This message indicates a system requirement; SYSOU1 must be BCD.

FILE ***** ASSOCIATED WITH REDUNDANT 'USE' STATEMENT. FIRST ONE RETAINED.
This message indicates that a redundant USE statement has been encountered. See the USE verb.

FILE ***** HAS NO ASSOCIATED FD ENTRY. ARBITRARY SPECIFICATIONS ASSUMED.
See the file description entry.

FILE ***** HAS NO RECORDS. INPUT/OUTPUT STATEMENT IGNORED.
See the DATA RECORDS clause of the file description entry.

FILE ***** IS ASSIGNED TO A CARD OR SYSTEM UNIT. OPTIONS SPECIFIED IN CLOSE STATEMENT ARE IGNORED.
This message indicates that certain system conventions (as indicated) take priority over COBOL options.

FILE ***** IS ASSIGNED TO ***** AND FILE RECORDING MODE IS BINARY. UNIT NOT PERMITTED TO BE CARD AT OBJECT-TIME.
This message indicates that system units SYSIN1, SYSOU1, and SYSPP1, should not be specified as card units at execution time. This is a system restriction.

FILE ***** IS ASSIGNED TO ***** AND MAXIMUM RECORD SIZE EXCEEDS 72 CHARACTERS. UNIT NOT PERMITTED TO BE CARD AT OBJECT-TIME.
This message indicates that system units SYSIN1, SYSOU1, and SYSPP1, should not be specified as card units at execution time. This is a system restriction.

***** FILE ***** IS ASSIGNED TO *****. UNIT NOT PERMITTED TO BE CARD AT OBJECT-TIME.
This message indicates that system units, SYSIN1, SYSOU1, and SYSPP1, should not be specified as card units at execution time. This is a system restriction.

***** FILE ***** NOT PERMITTED AS ARGUMENT IN 'USE' OPTION ***** STATEMENT.
See the USE verb.

FILE ***** RETENTION-PERIOD SPECIFICATION IGNORED SINCE ALLOWED ONLY FOR OUTPUT FILE.
See the VALUE clause of the file description entry.

FILE ***** SPECIFIED AS ***** INPUT ***** OUTPUT. ***** USAGE ASSUMED.
This message indicates that the rules governing the use of the USE verb have been violated. See the USE verb.

'FILLER' NOT PERMITTED AS FILE-NAME. FD ENTRY IGNORED.
Self-explanatory.

FIRST REPETITION OF SUBSCRIPTING ERROR. SUBSEQUENTLY, MESSAGES REFERRING TO SUBSCRIPTS APPEAR ONLY ONCE CORRESPONDING TO THE FIRST APPEARANCE OF EACH UNIQUE SUBSCRIPT DATA-NAME OR EXPRESSION.
No further messages will be generated for this error.

FLOATING POINT UNDERFLOW CONVERTING LITERAL. ZERO USED.
This message indicates the exponent became less than the minimum exponent which is 2^{-128} (approximately 10^{-38}).

FLOATING POINT OVERFLOW IN CONVERTING LITERAL. MAXIMUM VALUE USED.
This message indicates the exponent became greater than the maximum exponent which is 2^{+127} (approximately 10^{+38}).

***** FORCED TO LEVEL NUMBER OF 01.
Every data organization must have the highest order at the 01 level. See level-numbers in the DATA DIVISION.

***** ***** FROM ***** ***** TO ***** *****. ***** *****
See the MOVE verb.

'FROM' MAY ONLY BE FOLLOWED BY IBJOB STANDARD MNEMONIC-NAME 'SYSIN1.' SYSIN1 ASSUMED.
See the rules for the ACCEPT verb.

GROUP ITEM ***** USED AS A SUBSCRIPT. OBJECT PROGRAM USES SUBSCRIPT VALUE OF 1.
See subscripts.

***** HAS AN ILLEGAL LEVEL NUMBER. ASSUMED LEVEL NUMBER IS 49.
Valid level-numbers are 01 through 49, 77, and 88. If any other number is specified, 49 will be assumed. See level-numbers.

***** HAS AN ILLEGAL PICTURE. ***** IS ASSUMED PICTURE.
See the PICTURE clause of the data-item description.

***** HAS NO FILE DESCRIPTION. INPUT/OUTPUT STATEMENT IGNORED.
See the verb being used. Also check for proper working of the file description entry and the FILE-CONTROL paragraph.

***** HAS NOT APPEARED IN A SELECT ENTRY IN ENVIRONMENT DIVISION. FD ENTRY IGNORED.
Self-explanatory.

***** HAS NOT BEEN DEFINED IN A SELECT ENTRY AND IS IGNORED.
See the FILE-CONTROL paragraph of the INPUT-OUTPUT SECTION in the ENVIRONMENT DIVISION.

***** HAS VALUE CLAUSE TOGETHER ILLEGALLY WITH OCCURS OR REDEFINES. VALUE ACCEPTED IF OCCURS – APPLIED TO FIRST ELEMENT.
The OCCURS clause or the REDEFINES clause has been used illegally with the VALUE clause. See the three clauses in the data-item description entry.

HYPHENATED FORM OF ***** DESIGNATION PREFERRED.
See the CONFIGURATION SECTION of the ENVIRONMENT DIVISION.

I-O-CONTROL OPTIONS FOR ***** IGNORED SINCE ALLOWED ONLY FOR BINARY FILES.
This message reflects a system (IOCS) requirement.

I-O-CONTROL PARAGRAPH NOT FOLLOWING FILE-CONTROL PARAGRAPH IGNORED.
See "Organization of Source Program" and the ENVIRONMENT DIVISION.

ILLEGAL ***** UNIT ASSIGNED TO FILE ***** *****
See the FILE-CONTROL paragraph in the INPUT-OUTPUT SECTION of the ENVIRONMENT DIVISION.

ILLEGAL ARGUMENT IN 'ON' STATEMENT. STATEMENT DELETED.
Refer to "Debugging Package."

ILLEGAL ARITHMETIC PHRASE, ENDING WITH AN OPERATOR OTHER THAN RIGHT PARENTHESIS. PHRASE DELETED.
See formulas in the COMPUTE statement of the PROCEDURE DIVISION.

ILLEGAL CHARACTER IN COLUMN 7. SPACE IS ASSUMED.
A hyphen or a blank are the only characters allowed in column 7. See "Reference Format."

ILLEGAL CHARACTER IN LITERAL. CHARACTER IGNORED.
This message indicates one of the basic rules has been violated. See literals.

ILLEGAL CHARACTER ON CARD DELETED.
Self-explanatory.

ILLEGAL CLAUSE(S) DESCRIBING ***** IGNORED. LENGTH 1, VALUE OF R.M. ASSIGNED BY COMPILER. ONLY SYNCHRONIZATION, IF ANY, RETAINED.
Consult the specific format for the record mark (PICTURE J).

ILLEGAL CLAUSE(S) IN DESCRIPTION OF FLOATING POINT ITEM ***** IGNORED.
See the discussion on the specific format for floating-point items in the USAGE clause of the data-item description entry.

ILLEGAL CLAUSE(S) IN DESCRIPTION OF SCIENTIFIC DECIMAL ITEM ***** IGNORED.
See the discussion on the specific format for scientific decimal items in the PICTURE clause of the data-item description entry.

ILLEGAL CONDITIONAL EXPRESSION IN AT END OR ON SIZE ERROR CLAUSE.
This message has been generated because of a COBOL language restriction, but the 7090/7094 compiler accepts the statement. See the description of the AT END clause of the READ verb or of the ON SIZE ERROR clause relating to the ADD, SUBTRACT, MULTIPLY, or DIVIDE verb.

ILLEGAL CONDITIONAL EXPRESSION IN TEXT. EXPRESSION IGNORED.
See conditional statements in the PROCEDURE DIVISION.

ILLEGAL CONTROL SECTION NAME FOR DEBUG REQUEST. NAME IGNORED.
Refer to "Debugging Package."

ILLEGAL DESIGNATION OF SIGN CONVENTION IN PICTURE OF REPORT ITEM *****. +++++++ IS ASSUMED.
See the report form of the PICTURE clause of the data-item description entry.

ILLEGAL FORM OF VALUE FOR ***** *****. VALUE IGNORED.
Self-explanatory.

ILLEGAL INSERTION POINT SPECIFICATION FOR DEBUG REQUEST. REQUEST WILL NOT BE EXECUTED.
Refer to "Debugging Package."

ILLEGAL MIXTURE OF DIGIT POSITION CHARACTERS (9 Z *) AFTER POINT IN PICTURE OF REPORT ITEM *****. +++++++ IS ASSUMED PICTURE.
See the report form of the PICTURE clause of the data-item description entry.

ILLEGAL MIXTURE OF ORDER OF DIGIT POSITION CHARACTERS IN PICTURE OF REPORT ITEM *****. +++++++ IS ASSUMED PICTURE.
See the report form of the PICTURE clause of the data-item description entry.

ILLEGAL PICTURE OF SCIENTIFIC DECIMAL ITEM *****. +99999999E+99 IS ASSUMED PICTURE.
See the scientific decimal form of the PICTURE clause of the data-item description entry.

ILLEGAL PICTURE (OR NEITHER PICTURE NOR LEGAL SIZE GIVEN) FOR ***** DECIMAL ITEM *****. 999999 IS ASSUMED PICTURE.
The size must be specified by either the SIZE or the PICTURE clause.

ILLEGAL POINT OR SIGNED CLAUSE IN DESCRIPTION OF NON-REPORT DISPLAY ITEM *****.
See the data-item description entry in the DATA DIVISION.

ILLEGAL REDEFINITION IGNORED FOR FILE RECORD (01 LEVEL) NAMED *****.
The REDEFINES clause cannot be used with logical records (01 level) associated with the same file. See the REDEFINES clause of the data-item description entry.

ILLEGAL SENTENCE STRUCTURE. NOTHING DONE.
See punctuation.

ILLEGAL SUBSCRIPT STRUCTURE. SCAN RESUMED AT NEXT VERB, PERIOD, OR INFORMATION IN THE A MARGIN.
See subscripts.

ILLEGAL USAGE OR CLASS CLAUSE (OR BLANK WHEN ZERO) IN DESCRIPTION OF ALPHANUMERIC ITEM *****. CLAUSE IGNORED IN FAVOR OF PICTURE.
This warning message indicates a violation of a language rule.

ILLEGAL USAGE OR CLASS CLAUSE(S) IGNORED IN FAVOR OF PICTURE OF REPORT ITEM *****.
The PICTURE clause overrides contradictory USAGE and CLASS clauses.

ILLEGAL USE OF COMMA IN PICTURE OF REPORT ITEM *****. +++++++ IS ASSUMED PICTURE.
Self-explanatory.

ILLEGAL USE OF $ IN PICTURE OF REPORT ITEM *****. +++++++ IS ASSUMED PICTURE.
See replacement characters under the PICTURE clause of the data-item description entry.

ILLEGAL USE OF UNALTERABLE 'GO TO' STATEMENT 'GO TO' STATEMENT DELETED.
See the GO TO and ALTER verbs.

ILLEGAL USE OF V OR POINT IN PICTURE OF REPORT ITEM *****. +++++++ IS ASSUMED PICTURE.
See the report form of the PICTURE clause of the data-item description entry.

IMPROPER CHARACTER INTERRUPTS STRING OF + OR − OR $ IN PICTURE OF REPORT ITEM *****. +++++++ IS ASSUMED PICTURE.
See the floating +, −, or $ in the report form of the PICTURE clause of the data-item description entry.

IMPROPER LABEL CLAUSE IGNORED. COMPILER ASSUMES LABEL RECORD(S) OMITTED UNLESS VALUE CLAUSE PRESENT.
See the LABEL clause of the file description entry.

IMPROPER RECORDING CLAUSE IGNORED. BCD, HIGH DENSITY ASSUMED.
See the RECORDING MODE clause of the file description entry.

***** IN THE ENVIRONMENT DIVISION MUST NOT BE REPEATED. SCAN RESUMED AT NEXT PARAGRAPH, SECTION, OR DIVISION.
See "Organization of Source Program" and the ENVIRONMENT DIVISION.

IMCOMPLETE STATEMENT DELETED.
This message indicates invalid sentence structure. Consult the rules regarding the particular verb.

INELIGIBLE DATA-NAME ***** IN RECEIVE OR PROVIDE STATEMENT. SCAN RESUMED AT NEXT VERB, PERIOD, OR INFORMATION IN THE A-MARGIN.
See the ENTER verb.

INELIGIBLE DATA-NAME CANNOT BE USED AS AN ARGUMENT FOR THE CORRESPONDING OPTION.
See the CORRESPONDING clause under the MOVE verb.

'INPUT' or 'OUTPUT' MUST FOLLOW VERB IN AN 'OPEN' STATEMENT. STATEMENT DELETED.
The OPEN statement requires that either INPUT or OUTPUT be specified. See the OPEN verb.

INPUT/OUTPUT STATEMENT IGNORED.
See the PROCEDURE DIVISION discussion on the associated verb often associated with errors concerning SELECT or FD entries.

INTEGER MUST NOT EXCEED 32767. INTEGER 1 ASSUMED.
This message indicates that core storage capacity has been exceeded.

INTERNAL FILE ERROR
- INCONSISTENT FILE AND SERIAL
- PERMANENT CLOSE
- UNABLE TO STASH
- READOUT SEQUENCE
- UTILITY READ
- UTILITY EOF

Possible causes:
1. "short" of faulty utility tapes
2. oversize program
3. machine error

INVALID LITERAL USED IN EXAMINE STATEMENT.
Self-explanatory.

***** IS A NAME DEFINITIOIN AND MUST NOT BE QUALIFIED. DEFINITION FORCED.
This message refers to a data-name, condition-name, or procedure-name that was not properly defined; see the statement being used.

***** IS A TYPE OF ELEMENTARY DATA ITEM THAT MAY NOT BE USED AS A SUBSCRIPT. OBJECT PROGRAM USES SUBSCRIPT VALUE OF 1.
See subscripts.

***** IS A TYPE OF ELEMENTARY DATA ITEM THAT MAY NOT BE USED IN 'RETURN.' STATEMENT DELETED.
See the ENTER verb.

***** IS AN OUT-OF-RANGE REFERENCE.
Refer to "Debugging Package."

***** IS AN UNRECOGNIZABLE ITEM ON CARD. COMPILER SKIPS TO NEXT DIVISION.
The remainder of division is skipped. Check for spelling error.

***** IS GREATER THAN *****. FIRST VALUE USED IN DETERMINING MAXIMUM ***** SIZE.
See the file description entry.

***** IS IMPROPERLY QUALIFIED. DEFINITION FORCED.
See subscripts.

***** IS IMPROPERLY QUALIFIED, NAME IS NOT UNIQUE. DEFINITION FORCED.
This message refers to a data-name, condition-name, or procedure-name that was not properly defined; see qualification of names.

***** IS NOT *****. INPUT/OUTPUT STATEMENT IGNORED.
The READ verb requires a file-name; the WRITE verb requires a record name. See the rules regarding the particular verb.

***** IS NOT A FILE NAME. FD ENTRY IGNORED.
There is no legitimate SELECT entry in FILE-CONTROL paragraph. See the ENVIRONMENT DIVISION and the FILE SECTION of the DATA DIVISION.

***** IS NOT A FILE-NAME. I-O-CONTROL CLAUSE IGNORED.
A spelling error may have occurred in the file-name and no match can be found. The I-O-CONTROL paragraph in the INPUT-OUTPUT SECTION of the ENVIRONMENT DIVISION is ignored.

***** IS NOT A LEGAL CONDITION-NAME. REMAINDER OF SWITCH-NAME ENTRY IGNORED.
See the SPECIAL-NAMES paragraph in the CONFIGURATION SECTION of the ENVIRONMENT DIVISION.

***** IS NOT A LEGAL FILE-NAME. SELECT ENTRY IGNORED.
A language rule has been violated. See the FILE-CONTROL paragraph in the INPUT-OUTPUT SECTION of the ENVIRONMENT DIVISION.

***** IS NOT A LEGAL MNEMONIC-NAME. REMAINDER OF SWITCH-NAME ENTRY IGNORED.
See the SPECIAL-NAMES paragraph in the CONFIGURATION SECTION of the ENVIRONMENT DIVISION.

***** IS NOT A LITERAL. CLAUSE IGNORED.
See the VALUE clause of the data-item description entry.

***** IS NOT A NUMERIC LITERAL AND IS IGNORED.
See the VALUE clause of the file description entry.

***** IS NOT A PROCEDURE NAME. TRANSFER BYPASSING THIS STATEMENT INSERTED.
Incorrect reference. See the structure of the PROCEDURE DIVISION.

***** IS NOT DEFINED. DEFINITION FORCED UNLESS A QUALIFIER.
This message refers to a data-name, condition-name, or procedure-name that was not properly defined; see qualification.

***** IS STRUCTURALLY INCORRECT AT THIS POINT. I-O-CONTROL CLAUSE IGNORED.
A section head has been omitted. See "Organization of Source Program" and the ENVIRONMENT DIVISION.

***** IS STRUCTURALLY INCORRECT AT THIS POINT. REMAINDER OF SWITCH-NAME ENTRY IGNORED.
See the SPECIAL-NAMES paragraph in the CONFIGURATION SECTION of the ENVIRONMENT DIVISION.

***** IS STRUCTURALLY INCORRECT AT THIS POINT. SCAN RESUMED AT BEGINNING OF NEXT RERUN CLAUSE, PERIOD, OR PROPER INFORMATION IN THE A MARGIN.
See I-O-CONTROL paragraph of the INPUT-OUTPUT SECTION of the ENVIRONMENT DIVISION.

***** IS STRUCTURALLY INCORRECT AT THIS POINT. SCAN RESUMED AT BEGINNING OF NEXT SELECT ENTRY, PERIOD, OR PROPER INFORMATION IN THE A MARGIN.
See the FILE-CONTROL paragraph in the INPUT-OUTPUT SECTION of the ENVIRONMENT DIVISION.

***** IS STRUCTURALLY INCORRECT AT THIS POINT. SCAN RESUMED AT BEGINNING OF NEXT SWITCH-NAME ENTRY, PERIOD, OR PROPER INFORMATION IN THE A MARGIN.
See "Organization of the Source Program" and the ENVIRONMENT DIVISION.

***** IS STRUCTURALLY INCORRECT AT THIS POINT. SCAN RESUMED AT NEXT VERB, PERIOD, OR INFORMATION IN THE A MARGIN.
This message indicates that a language rule has been violated. See the PROCEDURE DIVISION information for the particular verb used.

***** IS UNIDENTIFIABLE. CLAUSE IGNORED.
See the DATA DIVISION.

***** IS UNIDENTIFIABLE. REMAINDER OF CLAUSE IGNORED.
See the FILE SECTION of the DATA DIVISION. This wording may also concern the PROCEDURE DIVISION; see message below.

***** IS UNIDENTIFIABLE. REMAINDER OF CLAUSE IGNORED.
Check spelling. Also study the rules of the particular verb used in this clause. This wording may also concern the FILE SECTION of the DATA DIVISION; see message above.

***** IS UNRECOGNIZABLE IN PROBABLE MULTIPLE REEL OPTION. MULTIPLE REELS ASSUMED.
This message indicates a probable spelling error.

***** IS UNRECOGNIZABLE. SCAN RESUMED AT NEXT DATA DESCRIPTION ENTRY, SECTION, OR DIVISION.
See the DATA DIVISION.

***** IS UNRECOGNIZABLE. SCAN RESUMED AT NEXT PARAGRAPH, SECTION, OR DIVISION.
See the PROCEDURE DIVISION.

ITEM ***** HAS NO SPECIFIED LENGTH. CONDITION IGNORED.
Length of a data-item must be specified. See the PICTURE and SIZE clauses of the data-item description entry.

JUSTIFIED CLAUSE IN DESCRIPTION OF ***** IGNORED. THIS FEATURE NOT IMPLEMENTED.
The JUSTIFIED clause is not a feature of 7090/7094 COBOL.

LABEL CLAUSE OMITTED IN FD ENTRY *****. COMPILER ASSUMES LABEL RECORD(S) OMITTED.
See the LABEL clause of the file description entry.

LENGTH OF NON-NUMERIC LITERAL EXCEEDS LENGTH SPECIFIED BY SIZE OR PICTURE CLAUSE FOR ***** LOW ORDER TRUNCATION DONE.
The alphanumeric constant contained within the VALUE clause should not be greater in size than the item. When it is greater, the low-order portion is truncated.

LENGTH OF ***** NOT BOTH CONSTANT AND LESS THAN 73 CHARACTERS. NOTHING DONE.
See the ACCEPT verb.

LENGTH (***** CHARACTERS) OF REDEFINING DATA FIELD HEADED BY DATA-NAME ***** IS GREATER THAN LENGTH (***** CHARACTERS) OF DATA FIELD BEING REDEFINED. DANGEROUS CONDITION IGNORED.
See the REDEFINES clause of the data-item description entry.

LEVEL FD SHOULD APPEAR IN THE A MARGIN. A MARGIN ASSUMED.
This message indicates a violation of a language rule. See "Reference Format."

LEVEL OF ***** CONFLICTS WITH THE PRECEDING LEVEL NUMBER CONDITION IGNORED.
See level-numbers in the data-item description entry.

LEVEL 77 ITEM ***** MAY NOT HAVE OCCURS.
See independent items (level-numbers 77) in the WORKING-STORAGE and CONSTANT SECTIONS.

LEVEL 77 ITEM ***** APPEARS IN FILE SECTION. INVALID DATA ORGANIZATION RESULTS.
Level 77 items are allowed only in the WORKING-STORAGE and CONSTANT SECTIONS.

LEVEL 88 CONDITION ***** LACKS MANDATORY VALUE CLAUSE.
Condition-names (level-number 88) must have a VALUE clause. See condition-names.

LEVEL 88 CONDITION ***** APPEARS ILLEGALLY in CONSTANT SECTION.
Condition-names (level-number 88) have no meaning in the CONSTANT SECTION. See condition-names.

LEVEL 88 CONDITION ***** NOT PRECEDED BY VALID ELEMENTARY ITEM.
See condition-names.

LIMIT (15 BITS) OF SIZE FIELD IN DICTIONARY NECESSITATES TREATING OPERATION LENGTH OF ***** AS MODULO 32768.
This message indicates a compiler limitation has been reached.

LITERAL FOLLOWING ALL IS LIMITED TO ONE CHARACTER. THE FIRST LITERAL CHARACTER IS USED.
See figurative constants.

***** LITERAL IS TOO LONG. FIRST ***** CHARACTERS WILL BE USED.
See the VALUE clause of the file description entry.

LOCATION FIELD FORMAT ERROR.
A numeric deckname was used on the $IBCBC card. This is a MAP message referring to the initial SAVE card of the assembly. If any other MAP message is produced, a systems engineer should be notified.

MACHINE OR COMPILER ERROR. COMPILATION IS INCOMPLETE.
(1) This message may indicate a machine error. Notify a customer engineer. (2) This message may indicate a compiler error. Consult a systems engineer concerning the APAR procedure.

MAXIMUM NUMBER ***** OF DIFFERENT NAMES IN A SOURCE PROGRAM EXCEEDED. COMPILATION TERMINATED.
This message indicates that a compiler limitation has been exceeded. Rework program into smaller, individual programs.

MAXIMUM RECORD SIZE (***** COMPUTER WORDS) EXCEEDS SPECIFIED BLOCK SIZE (***** COMPUTER WORDS) OF FILE *****. FILE IS SET UNBLOCKED.
This message indicates that the blocksize specified was not large enough. See the BLOCK CONTAINS clause of the file description entry.

MAXIMUM RECORD SIZE (***** COMPUTER WORDS) SPECIFIED IN FD ENTRY ASSOCIATED WITH FILE ***** IS NOT EQUAL TO SIZE OF MAXIMUM RECORD (***** COMPUTER WORDS) FD RECORD CLAUSE IGNORED.
This message indicates that there is an inconsistency between the record size specified in the file description entry and the record size given in the descriptions of the record. See RECORD CONTAINS and SIZE clauses in the FILE SECTION

..... ********** MESSAGE CAPACITY EXCEEDED.
This message indicates a compiler limitation. No more error messages can be generated. At least one further error has not been recorded.

MISUSE OF PERIOD, SIGN, OR E IN LITERAL. ILLEGAL CHARACTER(S) IGNORED.
This message indicates one of the mentioned rules has been violated. See literals.

MIXED CONTIGUOUS INSERTION-CHARACTERS IN PICTURE OF REPORT ITEM *****. +++++++ IS ASSUMED PICTURE.
See insertion characters under the PICTURE clause of the data-item description entry.

MNEMONIC ***** NOT UNIQUE, CONDITION IGNORED.
See the SPECIAL-NAMES paragraph in the CONFIGURATION SECTION of the ENVIRONMENT DIVISION.

MOVE FROM A FIGURATIVE CONSTANT TO A VARIABLE LENGTH GROUP ITEM NOT ALLOWED.
See the MOVE verb.

MOVE FROM A FIGURATIVE CONSTANT TO AN ITEM LONGER THAN 32767 CHARACTERS NOT ALLOWED.
This message indicates a compiler limitation has been reached. Suggest dividing data-item into smaller parts.

MULTIPLE ***** CLAUSES IN ***** DATA DESCRIPTION. FIRST ONE RETAINED.
This message indicates an extraneous clause has appeared. Only the first clause is retained.

MULTIPLE ***** CLAUSES IN FD ENTRY *****. FIRST ONE RETAINED.
See the file description entry.

MULTIPLE CONTIGUOUS INSERTION-CHARACTERS IN PICTURE OF REPORT ITEM ***** CHANGED TO A SINGLE CHARACTER.
This message indicates a compiler convention. See the report form of the PICTURE clause in the data-item description entry.

MULTIPLE REEL OPTION FOR FILE ***** OMITTED WHERE REQUIRED BUT IS ASSUMED.
MULTIPLE REEL must be specified if a file is on two or more reels of magnetic tape. See the FILE-CONTROL paragraph of the INPUT-OUTPUT SECTION in the ENVIRONMENT DIVISION.

NEITHER PICTURE NOR SIZE CLAUSE GIVEN FOR NON-REPORT DISPLAY ITEM *****. X IS ASSUMED PICTURE.
Self-explanatory.

NEITHER PICTURE NOR SIZE CLAUSE GIVEN FOR REPORT ITEM *****. +++++++ IS ASSUMED PICTURE.
See the data-item description entry.

NESTED REDEFINES ILLEGAL. REDEFINES CLAUSE IGNORED FOR *****.
Redefining is not allowed at a subordinate level to another REDEFINES. No more than one REDEFINES in one organization is permitted.

NO DIGIT POSITIONS IN PICTURE OF REPORT ITEM *****. ++++++ IS ASSUMED PICTURE.
See the report form of the PICTURE clause in the data-item description entry.

NO ERRORS WERE DETECTED BY THE COMPILER.
Self-explanatory.

NO PARAGRAPH NAME FOUND PRECEDING 'EXIT' STATEMENT. CONDITION IGNORED.
EXIT must always be a one-word paragraph. See the discussion of the EXIT verb.

NO RECORD DESCRIPTION ENTRIES FOLLOW FD ENTRY *****.
See "Organization of Source Program" and the DATA DIVISION.

NON-ALPHABETIC LITERAL GIVEN FOR ALPHABETIC ITEM *****. CONDITION IGNORED.
The value given for an alphabetic item may not contain nonalphabetic characters.

NON-NUMERIC LITERAL CONTINUATION MUST BEGIN WITH A QUOTE. QUOTE ASSUMED PRECEDING FIRST NON-SPACE CHARACTER.
See "Reference Format."

NON-NUMERIC LITERAL LONGER THAN 120 CHARACTERS OR NAME LONGER THAN 30 CHARACTERS TRUNCATED.
This message indicates a language restriction. See literals.

NON-NUMERIC LITERAL VALUE OF NUMERIC ITEM ***** IGNORED.
Numeric items may have only numeric values. See the VALUE clause in the DATA DIVISION.

***** NOT A LABEL-DATA-NAME. REMAINDER OF VALUE CLAUSE IGNORED.
See the VALUE clause of the file description entry.

NOT IN 'DECLARATIVES' MODE. STATEMENT DELETED.
The USE verb may be used only in the declarative section. See the USE verb.

NOTE ... FILE-NAME CHANGED FOR INTERNAL PURPOSES TO *****.
Internal limitations make change of file-name mandatory. No information is lost.

NUMBER OF DIGITS IN FIELD OF LITERAL EXCEEDS LIMIT OF 18. 18 DIGITS USED.
A numeric literal may not contain more than 18 characters. See the rules for literals.

NUMBER OF FILES NAMED IN FILE-CONTROL EXCEEDS MAXIMUM OF 63. COMPILATION TERMINATED.
This message indicates that a compiler limitation has been reached. This is a D-level message causing compilation to stop. A dump also results. It is suggested that the job be broken into smaller jobs.

NUMBER OF OCCURRENCES OF ***** IS ILLEGAL. COMPILER ASSUMES OCCURS EXACTLY 100 TIMES.
See the OCCURS clause of the data-item description entry.

NUMBER OF OCCURRENCES OF ITEM ***** DEPENDS ON A FOLLOWING ITEM IN THE SAME DATA GROUP. 'DEPENDING ON' CLAUSE IGNORED.
The description of the item that determines the number of repetitions cannot follow the item with the OCCURS DEPENDING ON CLAUSE. See the OCCURS clause of the data-item description entry.

OBJECT-COMPUTER DESIGNATION OVERRIDDEN BY ***** OPTION ON $IBCBC CARD.
This message indicates that the number of index registers specified on the $IBCBC card does not agree with the number of index registers designated by the object-computer.

OCCURS CLAUSE IGNORED FOR CONDITIONAL VARIABLE *****.
See conditional variables.

OCCURS CLAUSE NOT PERMITTED FOR HIGHEST LEVEL DATA ITEM *****.
This message indicates a language limitation. An OCCURS clause may not be used at 01 level.

***** OF FILE ***** ASSIGNED TO ***** NOT PERMITTED.
This message indicates a system function error. Blocking is not permitted on system units.

***** OF GROUP ITEM ***** IGNORED DUE TO CONFLICT WITH A HIGHER ORDER GROUP.
Self-explanatory.

ONLY FILE-NAMES MAY BE USED AS ARGUMENTS IN 'OPEN' OR 'CLOSE' STATEMENTS. STATEMENT DELETED.
See OPEN and CLOSE statements in the PROCEDURE DIVISION.

OPERAND TABLE OVERFLOW TRANSLATING EXPRESSION. STATEMENT DELETED.
The statement is too large. Re-arrange the statement in smaller parts.

OPERATION IGNORED BECAUSE ***** HAS IMPROPER DATA FORMAT.
See the rules regarding the particular verb.

OPERATION IGNORED BECAUSE ILLEGAL USE OF FIGURATIVE CONSTANT.
See figurative constants.

***** ORDER TRUNCATION OCCURS IN GENERATION OF INITIAL VALUE FOR *****.
See VALUE clause of the data-item description entry.

***** PARAGRAPH APPEARS ILLEGALLY IN ***** SECTION. SCAN RESUMED AT NEXT PARAGRAPH, SECTION, OR DIVISION.
See "Organization of Source Program" and ENVIRONMENT DIVISION.

PERFORM STATEMENT STRUCTURALLY INCORRECT. STATEMENT DELETED.
See the PERFORM verb.

PERMANENT READ ERROR DURING PROCEDURE INSTRUCTION GENERATION PHASE. COMPILATION IS SUSPECT.
This message indicates a bad utility tape. Do not trust compilation. Suggest re-compilation.

PICTURE OF ALPHANUMERIC ITEM ***** CONTAINS A MIXTURE OF A'S AND 9'S – TREATED AS ALL X'S.
This message indicates a language rule violation.

PICTURE OF REPORT ITEM ***** HAS ILLEGAL USE OF SCALING CHARACTER P. +++++++ IS ASSUMED PICTURE.
See the PICTURE clause of the data-item description entry.

PICTURE OF REPORT ITEM ***** HAS SCALING CHARACTER P EMBEDDED ILLEGALLY BETWEEN NUMERIC CHARACTER POSITIONS +++++++ IS ASSUMED PICTURE.
See the PICTURE clause of the data-item description entry.

PICTURE OF REPORT ITEM ***** IS ILLEGAL. +++++++ IS ASSUMED PICTURE.
See the report form of the PICTURE clause in the data-item description entry.

PREVIOUS DATA DESCRIPTION NOT TERMINATED BY A PERIOD. PERIOD ASSUMED AND PROCESSING OF ***** BEGUN.
Self-explanatory.

PRIMARY AND SECONDARY UNITS ASSIGNED TO FILE ***** CONFLICT. SECONDARY UNIT ASSIGNMENT IGNORED.
Self-explanatory.

PROCEDURE STATEMENT TO ***** FILE ***** NOT GIVEN.
Every file named in a SELECT statement in the FILE-CONTROL paragraph of the ENVIRONMENT DIVISION must be opened and closed in the PROCEDURE DIVISION. See the verbs OPEN and CLOSE.

QUANTITY ITEM ***** SHOULD NOT BE USED WITH 'REPLACING' OPTION IN 'EXAMINE' STATEMENT. CONDITION IGNORED.
See the EXAMINE verb.

RECORD ***** APPEARS IN RECORD DESCRIPTION ENTRY BUT WAS NOT LISTED IN THE FD DATA RECORD(S) CLAUSE. CONDITION IGNORED AND RECORD DESCRIPTION RETAINED.
See the DATA RECORDS clause of the file description entry.

RECORD HEADED BY DATA ITEM ***** EXCEEDS 32767 WORDS. LENGTH MODULO 32768 USED.
This message indicates a compiler limitation has been reached.

RECORD ***** LISTED IN THE FD DATA RECORD(S) CLAUSE DOES NOT APPEAR IN A RECORD DESCRIPTION ENTRY. CONDITION IGNORED.
See the DATA RECORDS clause of the file description entry.

REDEFINES DESCRIBING ***** NOT FOLLOWED BY A PREVIOUSLY DEFINED DATA-NAME. CLAUSE IGNORED.
See the REDEFINES clause of the data-item description entry.

REDUNDANCY ON SYSTEM INPUT UNIT. CONDITION IGNORED.
One card image may have been lost. Suggest re-compilation.

REDUNDANT FD ENTRY ***** IGNORED.
Self-explanatory.

REDUNDANT FD ENTRY ***** IGNORED. ONLY ONE FD ENTRY MAY DESCRIBE A SET OF RENAMED SELECT ENTRIES.
See description of RENAMING clause of the SELECT entry.

REDUNDANT I-O-CONTROL SPECIFICATION.
SEQUENCE-CHECK, CHECK-SUM, or RERUN has been specified more than one time for a given file in the I-O-CONTROL paragraph of the ENVIRONMENT DIVISION. For example, the following two cards in the same program cause check-sum to be specified for FILE-NAME-1 twice:

```
APPLY CHECK-SUM ON FILE-NAME-1.
APPLY CHECK-SUM ON ALL FILES.
```

The card whose number is given with the message contains the SELECT entry for the file involved.

REDUNDANT SELECT ENTRY ***** IGNORED.
A file-name has been selected more than once in the FILE-CONTROL paragraph of the ENVIRONMENT DIVISION.

REDUNDANT SWITCH-NAME ENTRY FOR KEY ***** IGNORED.
See the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION.

REDUNDANT 'USE' STATEMENT.
STATEMENT DELETED.
This message indicates that a redundant USE statement has been encountered. See the USE verb.

'RENAMING' MAY ONLY BE FOLLOWED BY A FILE-NAME, REMAINDER OF SELECT ENTRY IGNORED. ASSUMED UNIT ASSIGNMENT IS '1 TAPE-UNIT'
If the RENAMING option is used in the FILE-CONTROL paragraph of the ENVIRONMENT DIVISION, it must be followed by a file-name.

SECTION-HEADER ***** SECTION NOT FOLLOWED BY ***** DESCRIPTION ENTRY. SCAN RESUMED AT NEXT ***** DESCRIPTION ENTRY, SECTION, OR DIVISION.
See "Organization of Source Program."

SECTION IN THE DATA DIVISION MUST ***** SCAN RESUMED AT NEXT SECTION, OR DIVISION.
See "Organization of Source Program."

SENTENCE LENGTH EXCEEDS COMPILER CAPACITY. SUGGEST SUBDIVIDING SENTENCE INTO SMALLER COMPONENTS.
Self-explanatory.

SEQUENCE-CHECK MUST BE SPECIFIED WHEN CHECK-SUM IS SPECIFIED. SEQUENCE-CHECK ASSUMED.
This message reflects a system (IOCS) requirement.

729-MODEL NO. ***** ASSIGNED TO FILE *****, NOT ACCEPTABLE TO LOADER, CHANGED BY COMPILER TO *****.
This message indicates a system restriction. Certain 729 Magnetic Tape Units cannot be used.

***** SHOULD BE FOLLOWED BY A SPACE. SPACE IS ASSUMED.
See punctuation.

***** SHOULD NOT BE FOLLOWED BY A SPACE. CONDITION IGNORED.
See punctuation.

***** SHOULD NOT BE IN THE A MARGIN. B MARGIN ASSUMED.
See "Reference Format."

SIZE OF ***** GIVEN AS 0. COMPILER ASSUMES SIZE IS 6.
Zero is an illegal size.

SIZE, POINT, SIGNED, OR EDITING CLAUSES IGNORED IN FAVOR OF PICTURE IN *****.
This message indicates that some information has been duplicated. In this case, the PICTURE clause information is contrary to similar information specified in another clause. The PICTURE is correct.

SOURCE-COMPUTER IS IMPROPERLY SPECIFIED. IBM-7090 ASSUMED.
The hyphen must be specified in "IBM-7090" in the CONFIGURATION SECTION of the ENVIRONMENT DIVISION.

***** SPECIFICATION ***** REMAINDER OF VALUE CLAUSE IGNORED.
See the VALUE clause of the file description entry.

***** SPECIFICATION IN ***** CLAUSE NOT AN UNSIGNED INTEGER. CLAUSE IGNORED.
See the DATA DIVISION.

***** SPECIFICATION IN ***** CLAUSE NOT AN UNSIGNED INTEGER. REMAINDER OF CLAUSE RETAINED.
See the DATA DIVISION.

STATEMENT CONTAINS TOO FEW RIGHT PARENTHESES. COMPENSATING PARENTHESES ADDED AT END OF STATEMENT.
The number of right parentheses must equal the number of left parentheses.

STATEMENT CONTAINS TOO MANY RIGHT PARENTHESES. EXTRA PARENTHESES IGNORED.
The number of right parentheses must equal the number of left parentheses.

STATEMENT REQUIRES A DATA-NAME, LITERAL, OR FIGURATIVE CONSTANT, NOT *****, AS AN ARGUMENT, STATEMENT DELETED.
See the rules regarding the use of the particular verb referred to in this message.

STATEMENT REQUIRES A PROCEDURE NAME, NOT *****, AS AN ARGUMENT. STATEMENT DELETED.
See the rules regarding the use of the particular verb referred to in this message.

SUBORGANIZATION OF ITEM ***** WITH OCCURS CLAUSE CONTAINS A VALUE CLAUSE. VALUE GIVEN TO FIRST ELEMENT ONLY.
A VALUE clause cannot appear in a description subordinate to one containing an OCCURS clause. See the VALUE and OCCURS clauses of the data-item description entry.

SUBSCRIPT COUNT EXCEEDS 3. SCAN RESUMED AT NEXT VERB, PERIOD, OR INFORMATION IN THE A MARGIN.
Only three levels of subscripting are allowed. See subscripts.

SUBSCRIPT INTEGER MUST NOT EXCEED 32767. INTEGER 1 ASSUMED.
This message indicates that a compiler limitation has been exceeded.

SUBSCRIPT MISSING AFTER LEFT PARENTHESIS. SCAN RESUMED AT NEXT VERB, PERIOD, OR INFORMATION IN THE A MARGIN.
Information has been omitted.

SYNCHRONIZED ITEM ***** HAS REDEFINES CLAUSE. STORAGE ASSIGNMENT MIGHT NOT BEGIN WITH THE FIRST CHARACTER POSITION OF THE REDEFINED AREA. CONDITION IGNORED.
This message indicates the characteristics of the REDEFINES clause take priority.

SYSIDR ACCOUNTING ROUTINE ERROR.
Contact your systems engineer whenever this message occurs.

T2 WORD – ********** L(TSX) – ***** L(SKEL) – *****
Consult your systems engineer concerning APAR procedure whenever this message occurs. It is always accompanied by the MACHINE OR COMPILER ERROR message.

TERMINATION OF LITERAL FORCED AT END OF CARD.
This message indicates that either a quote mark or a continuation mark has been omitted. See literals.

TOO FEW OPERANDS IN ADD STATEMENT. STATEMENT DELETED.
A minimum of two operands is allowed in an ADD statement.

TOO FEW SUBSCRIPTS GIVEN FOR ***** COMPILER ASSUMES MISSING LEFTMOST SUBCRIPTS TO BE 1.
See subscripting.

TRANSFER BYPASSED BECAUSE ***** IS NOT A STATEMENT OR SECTION NAME.
Self-explanatory.

21 PERMANENT READ REDUNDANCIES ON SYSTEM INPUT UNIT. COMPILATION TERMINATED.
This message indicates 21 card read errors. Compilation is terminated and a dump is taken.

UNIDENTIFIABLE WORD ***** IN DATA DESCRIPTION. WORD OR CLAUSE IGNORED.
This message indicates a program error. See the DATA DIVISION.

'UPON' MAY ONLY BE FOLLOWED BY IBJOB STANDARD MNEMONIC-NAME 'SYSOU1.' SYSOU1 ASSUMED.
SYSOU1 is the only unit permitted for DISPLAY statements.

UPSCALE MAY CAUSE HIGH ORDER TRUNCATION FOR STORE INTO *****.
High-order truncation may occur because of decimal point alignment in a receiving area that is too small. An error will probably result from this operation.

'USE' NOT PRECEDED BY SECTION-NAME.
See the rules regarding the USE verb in the declaratives section of the PROCEDURE DIVISION.

***** USED AS A SUBSCRIPT HAS AN ALPHANUMERIC PICTURE, BUT VALUES MUST BE RESTRICTED TO INTEGERS.
See subscripts.

***** USED AS A SUBSCRIPT HAS AN ALPHABETIC PICTURE. INVALID SUBSCRIPT. OBJECT PROGRAM USES SUBSCRIPT VALUE OF 1.
See subscripts.

***** USED AS SUBSCRIPT IS A SIGNED EXTERNAL DECIMAL ITEM. NEGATIVE VALUES CAUSE ERRORS.
Subscripts must have positive integer values. See subscripts.

***** USED AS A SUBSCRIPT IS IN BCD. OBJECT-TIME CONVERSION AND/OR UNPACKING IS REQUIRED FOR SUBSCRIPTS NOT COMPUTATIONAL, SYNCHRONIZED RIGHT.
In most cases computational, synchronized right items are more efficient as subscripts.

***** USED AS A SUBSCRIPT. IS INVALID DUE TO NONZERO SCALING. OBJECT PROGRAM USES SUBSCRIPT VALUE OF 1.
Subscripts must be positive integers. See subscripts.

***** USED AS A SUBSCRIPT. IS NOT SYNCHRONIZED RIGHT. OBJECT-TIME UNPACKING IS REQUIRED.
In most cases synchronized right items are more efficient as subscripts.

*****, USED TO CONTROL A GO TO, HAS ILLEGAL FORMAT. GO TO STATEMENT IGNORED.
See the GO TO DEPENDING ON statement in the PROCEDURE DIVISION.

*****, USED TO CONTROL A GO TO, IS NOT AN INTEGER. INTEGER PART USED.
See the GO TO DEPENDING ON statement in the PROCEDURE DIVISION.

'USING' MUST BE FOLLOWED BY THE NAME OF A FIXED-LOCATION DATA-ITEM. STATEMENT DELETED.
See the description of the CALL form of the ENTER verb.

VALUE CLAUSE OF ITEM ***** IGNORED SINCE IT IS EITHER REDEFINED, PRECEDED BY A VARIABLE LENGTH ITEM, OR IS IN THE FILE SECTION.
This message indicates that certain clauses are contradictory. See the VALUE clause of data-item description.

VALUE CLAUSE OMITTED IN FD ENTRY *****. LEGAL BUT UNUSUAL.
See the VALUE clause of the file description entry.

WARNING. LISTING OF THIS NAME HAS BEEN TERMINATED. PROGRAM IS NOT AFFECTED.
The REF has been used. Because the number of items cross-referenced exceeds compiler limitations, the REF list may be incomplete.

***** WITH REDEFINES CLAUSE NOT IMMEDIATELY PRECEDED BY THE REDEFINED AREA REDEFINES IGNORED.
The redefining item must follow the redefined item with no intervening entries. See the REDEFINES clause of the data-item description entry.

'WITHOUT' MUST BE FOLLOWED BY COBOL WORDS 'COUNT CONTROL' IN RECORDING CLAUSE. NOTHING DONE.
See the RECORDING MODE clause of the file description entry.

WORDING 'EVERY BEGINNING OF REEL' PREFERRED IN RERUN CLAUSE AND IS ASSUMED.
This message indicates obsolete wording. See the I-O-CONTROL paragraph of the ENVIRONMENT DIVISION.

NOTE: If a compilation is terminated and a dump taken with no error message being printed, a systems engineer should be notified.

NOTE: The MAP message LOCATION FIELD FORMAT ERROR, which refers to the initial SAVE card of the assembly, will be given if a numeric deckname is used on the $IBCBC card. If any other MAP message is produced, a systems engineer should be notified.

Appendix A

Reference Format

A COBOL program may be written on COBOL program sheets. The rules for using the program sheet are given below.

Contents of Columns

Columns 1-6: These columns may be used for sequence numbers. The first three columns give the page number and the next three columns give the line number. Only numbers, not letters of the alphabet or special characters, may be used in these columns. Sequence numbers are optional, but they are checked if they are present.

Column 7: This column is used only to indicate the continuation of a word to a new line. A hyphen in column 7 signifies that the first character on the new line is to follow the last character on the preceding line without an intervening space (no other character may appear in column 7). An alphanumeric literal (one or more words enclosed in quotation marks) constitutes one COBOL word. If the literal is continued from one line to the next, a hyphen must be put in column 7 of the new line. Remaining spaces on the first line are considered to be part of the literal. A quotation mark must precede the portion of the literal that is on the new line but leading spaces before the quotation mark are not considered to be part of the literal.

Columns 8-72: These columns are used for the program. Formats for the entries are given throughout this publication.

Columns 73-80: These columns are used for identification. They may contain any character from the character set and have no effect on the program.

Margins

Margin A (columns 8-11) and margin B (columns 12-72) are marked on the program sheet. The entries that must begin in each margin are:

1. The names of divisions must begin at margin A, followed by a space and the word DIVISION, followed by a period. No other information can appear on the line.

2. The names of sections must begin at margin A, followed by a space and the word SECTION, followed by a period.

3. Paragraph names must begin at margin A and be immediately followed by a period and a space. Succeeding lines in the paragraph must begin in margin B.

4. File description entries must begin in margin A. Succeeding lines in the file description entry must begin in margin B.

5. Data-item descriptions may begin in margin A or B, but a data-name cannot begin before margin B.

Punctuation

The punctuation given in the entry formats in this publication must be used. The following list gives the rules for punctuation.

1. A period or comma must immediately follow a word and must be followed immediately by a space. A period must follow a division-name, a section-name, a paragraph-name, a complete file description or data-item description entry, and a complete sentence or paragraph. The use of commas is optional, they may be used to separate a series of items.

2. At least one space must appear between two successive words and/or parenthetical expressions. Two or more successive spaces are treated as a single space except in nonnumeric literals.

3. A beginning quotation mark must not be followed by a space and an ending quotation mark must not be preceded by a space, unless the spaces are desired in a nonnumeric literal. All spaces within the quotation marks for nonnumeric literals are included in the literal. Nonnumeric literals and entry-names (in an ENTER statement) must be enclosed in quotation marks.

4. Punctuation rules are suspended inside the quotation marks for nonnumeric literals.

5. A left parenthesis must not be immediately followed by a space, and a right parenthesis must not be immediately preceded by a space. Subscripts must be enclosed in parentheses. Parentheses may be used to indicate the order of operations.

6. Arithmetic operators (and the equal sign) must be preceded and followed by a space.

Types of Names

The following types of names can be used in a COBOL program.

1. data-names: Names assigned to either group or elementary data-items in data-item description entries.

2. file-names: Names assigned to files in file description entries.

3. *condition-names:* Names assigned, in data-item description entries, to a specific value in the range of values a data-item may assume. Condition-names are identified by the special level-number 88.

4. *switch-status-names:* Names assigned to identify the entry keys. Switch-status-names are assigned in the ENVIRONMENT DIVISION (SPECIAL-NAMES paragraph) and may be tested with a switch-status test in the PROCEDURE DIVISION.

5. *entry-names:* Names used with ENTER verbs to identify entry points of subprograms.

6. *paragraph-names:* Names assigned to paragraphs in the PROCEDURE DIVISION.

7. *section-names:* Names assigned to sections in the PROCEDURE DIVISION.

8. *procedure-names:* Paragraph-names and section-names.

Name Formation

Names consist of a combination of the letters A through Z, the numbers 0 through 9, and the hyphen. The following rules must be followed in forming names.

1. Names must not contain blanks (spaces).
2. They may contain from 1 to 30 characters.
3. They may neither begin nor end with a hyphen, although hyphens may be used in the name for readability.
4. File-names, data-names, condition-names, entry-names, and switch-status-names must contain at least one alphabetic character. Paragraph-names and section-names may consist entirely of numbers.
5. Names need not be unique if they can be qualified.

Literals

A literal is a word or number that defines itself. Literals have constant values that do not vary in a program. Nonnumeric literals must be enclosed in quotation marks. Numeric literals must not be enclosed in quotation marks.

Nonnumeric literals may be composed of from 1 to 120 alphanumeric characters (except the quotation mark) counting embedded spaces as characters.

Numeric literals may be composed of from 1 to 18 characters chosen from the digits 0 through 9, the sign character, and the decimal point. There may be at most one sign and/or one decimal point. The sign must appear as the leftmost character (unsigned numbers are assumed to be positive) and the decimal point may appear anywhere except as the rightmost character.

Floating-point literals are a special type of numeric literal and are written in the following special floating-point format. There must be no embedded blanks.

$$\left\{\begin{matrix}+\\-\end{matrix}\right\} \text{mantissa} \quad \text{E} \left\{\begin{matrix}+\\-\end{matrix}\right\} \text{exponent}$$

The mantissa may contain from 1 to 16 digits and a decimal point (except in the rightmost position). The exponent may be one or two digits and the magnitude must not exceed 38. Floating-point literals may be used only in the DATA DIVISION. All other literals may be used in the PROCEDURE DIVISION.

Examples of nonnumeric literals are:

```
"EXAMINE CLOCK NUMBER"
"PAGE 144 MISSING"
"-125.65"
```

Examples of numeric literals are:

```
1506789
-1257.6
+258.65
+2.6E-16
```

Qualifiers

A qualifier is a name preceded by IN or OF that is used to make a subordinate name unique. For example, if two paragraphs in a program are named PARAGRAPH2, and one of them is in a section named SECTION2 and the other in a section named SECTION3, a unique reference can be made by specifying PARAGRAPH2 IN SECTION2 OR PARAGRAPH2 OF SECTION3. The following rules must be observed for qualification:

1. Subscripts, conditional variables, condition-names procedure-names, and data-names may be made unique by qualification.
2. A file-name is the highest level qualifier available for a data-name. A section-name is the only qualifier available for a paragraph-name. File-names and section-names must be unique since they cannot be qualified.
3. Names must appear in ascending order of hierarchy with either the word IN or OF separating them. Enough qualification must be mentioned to make the name unique.
4. A qualifier must be of a higher level and in the same hierarchy as the name it is qualifying.
5. The highest level qualifier must be unique.
6. The same name must not appear at two levels in a hierarchy.
7. Each name in a hierarchy must be unique at its own level. For example, in one group there cannot be two elementary items with identical names and level numbers.
8. Paragraph-names must not be duplicated in the same section. Paragraph-names need not be qualified when referred to within the section.

9. A data-name cannot be subscripted when it is being used as a qualifier.

10. A name can be qualified even though it does not need qualification. The use of more names for qualification than are required for uniqueness is permitted. If there is more than one combination of qualifiers that insures uniqueness, any set can be used.

Subscripts

A subscript is a value that locates a particular item in a set of repeated items. For example, if the clause OCCURS 5 TIMES is used in the data-item description of an item named NUMBER, the third occurrence could be referred to as NUMBER (3).

The following rules apply to the use of subscripts:

1. The subscript may be represented by a literal that is a positive integer, or by a data-name that has a positive integral value, or by the special register TALLY.

2. Subscripts are enclosed in parentheses and separated by commas.

3. If sets of repeated items are nested, the use of multiple subscripts to refer to a particular item is permitted.

4. A maximum of three levels of subscripts is permitted.

5. Multilevel subscripts are written from left to right in the order: major, intermediate, minor. A subscript value of 1 denotes the first element of a list, a value of 2 refers to the second element of the list, etc. A subscript with the value (1, 2) refers to the second element within the first set of repeated elements. The last item in a nested set of repeated items with the major element appearing ten times, the intermediate element appearing five times in each major element, and the minor element appearing three times in each intermediate element, has the subscript value (10, 5, 3).

6. The name of the minor element in a nested set of items is the one subscripted.

7. If a data-item is repeated, that is, described with an OCCURS clause or is a subdivision of an item described with an OCCURS clause, it must be subscripted when it is used.

8. A data-name can be subscripted only if there is an OCCURS clause in its data-item description or in the description of a group item to which it belongs.

9. If a data-item is described as OCCURS *positive-integer* TIMES DEPENDING ON *data-name*, the number of repetitions is determined by the value of *data-name* at the time reference is made to the repeated item. *Data-name* cannot have a negative or zero value at this time.

10. The same data-name can be used as a subscript for different items.

11. A data-name must not be subscripted when:
    a. it is being used as a subscript.
    b. it is being used as a qualifier.
    c. it is in the DATA DIVISION.

There are several correct ways of expressing subscripted data-names. For example, if a data-item named A occurs five times and contains a data item named B that occurs four times in each A, and each B, in turn, contains a data-item C that occurs twice in each B, then the following expressions are all correct references to the last C, that is, to the second C in the fourth B in the fifth A:

```
C IN B IN A (5, 4, 2)
C IN B (5, 4, 2)
C IN A (5, 4, 2)
C (5, 4, 2)
```

The following forms of expressions are incorrect:

```
C (5, 4, 2) IN B IN A
C (2) IN B (4) IN A (5)
C (4, 2) IN A (5)
C (2) IN B (5, 4)
```

Figurative Constants

A figurative constant is a value with a preassigned fixed data-name. It is not enclosed in quotation marks. The fixed data-names and their meanings are given below. The singular and plural forms may be used interchangeably.

ZERO ZEROS ZEROES	Represent one or more zeros (0).
SPACE SPACES	Represent one or more blanks or spaces.
HIGH-VALUE HIGH-VALUES	Usually represent one or more 9's, the highest value in the commercial collating sequence; but represent one or more left parentheses if the scientific (binary) collating sequence has been specified by the option BINSEQ on the $IBCBC control card.
LOW-VALUE LOW-VALUES	Usually represent one or more blanks or spaces, the lowest value in the commercial collating sequence; but represent one or more zeros if the scientific (binary) collating sequence has been specified by the option BINSEQ on the $IBCBC control card.
QUOTE QUOTES	Represent the quotation character '. Note that the use of the figurative constant QUOTE to represent the character ' is not equivalent to the use of the symbol ' to bound a literal.
ALL 'literal'	Represents one or more occurrences of *'literal'*. *'Literal'* represents a single character nonnumeric literal and must be enclosed in quotation marks. An alternate form of *'literal'* is any figurative constant except ALL, e.g., ALL SPACES.

ZERO (ZEROS, ZEROES) is the only figurative constant that may be used with a COMPUTATIONAL data-item. All

of the figurative constants may be used with DISPLAY data-items.

The following examples show uses of figurative constants:

1. MOVE ALL '4' TO COUNT-FIELD, where COUNT-FIELD has been described as having 6 characters, results in 444444.
2. MOVE SPACES TO TITLE-BOUNDARY results in the item TITLE-BOUNDARY being filled with spaces.
3. The statement DISPLAY QUOTE, 'NAME', QUOTE results in the word 'NAME' displayed on the DISPLAY device.
4. MOVE QUOTE TO AREA-A, where AREA-A has been described as having five characters, results in '''''.

Organization of Source Program

The following is a list of the items that may appear in a source program. Some of the items are required; others are optional. This may be checked in the discussion of the individual entries. The sequence of entries given here is recommended; the division must appear as shown.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
AUTHOR.
INSTALLATION.
DATE-WRITTEN.
DATE-COMPILED.
SECURITY.
REMARKS.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.
OBJECT-COMPUTER.
SPECIAL-NAMES.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
I-O-CONTROL.

DATA DIVISION.
FILE SECTION.
FD file-name-1
  01 data-name
    02 data-name
      03 data-name
        88 condition-name
          .
          .
    02 data-name
    .
    .
  01 data-name
  .
  .
FD file-name-n
.
.
WORKING-STORAGE SECTION.
77 data-name
  88 condition-name
  .
  .
77 data-name
01 data-name
  02 data-name
    03 data-name
      88 condition-name
        .
        .
  02 data-name
  .
  .
01 data-name
.
.
CONSTANT SECTION.
77 data-name
.
.
77 data-name
01 data-name
  02 data-name
    03 data-name
      .
      .
  02 data-name
  .
  .
01 data-name
.
.
PROCEDURE DIVISION.
DECLARATIVES.
section-name-1 SECTION.
paragraph-name-1.
        .
        .
paragraph-name-n.
        .
        .
section-name-n SECTION.
        .
        .
END DECLARATIVES.
section-name-1 SECTION.
paragraph-name-1.
        .
        .
paragraph-name-n.
        .
        .
section-name-n + m SECTION.
        .
        .
```

Appendix B

Standard Label Format

In a COBOL program, the VALUE clause of the file description entry is the only means of referring to the label area. An example of modifying a label using a MAP program is given in Part II of this publication.

Format of the IBM Standard 84-Character Header Label

```
01 STANDARD-HEADER-LABEL USAGE IS DISPLAY.
02 LABEL-IDENTIFIER PICTURE X(6).
88 HEADER VALUE '1HDR'.
02 FILLER PICTURE X.
02 REEL-SERIAL-NUMBER PICTURE X(5).
02 FILLER PICTURE X.
02 FILE-SERIAL-NUMBER PICTURE X(5).
02 FILLER PICTURE X.
02 REEL-SEQUENCE-NUMBER PICTURE X(4).
02 CREATION-DATE.
03 YEAR PICTURE 99.
03 FILLER PICTURE X.
03 DAY PICTURE 999.
02 FILLER PICTURE X(3).
02 RETENTION-DAYS PICTURE 9(3).
02 FILLER PICTURE X.
02 FILE-DENSITY PICTURE 9.
02 FILE-MODE PICTURE 9.
02 CHECK-SUM-INDICATOR PICTURE 9.
02 BLOCK-SEQUENCE-INDICATOR PICTURE 9.
02 CHECKPOINT-INDICATOR PICTURE 9.
02 FILE-IDENTIFICATION PICTURE X(18).
02 SORT-RESERVED PCTURE X(12).
02 OPTIONAL PICTURE X(12).
```

Format of the IBM Standard 84-Character Trailer Label

```
01 STANDARD-TRAILER-LABEL USAGE IS DISPLAY.
02 LABEL-IDENTIFIER PICTURE IS X(6).
88 END-OF-REEL VALUE IS '1EOR'.
88 END-OF-FILE VALUE IS '1EOF'.
02 FILLER PICTURE X.
02 BLOCK-COUNT PICTURE 9(5).
02 FILLER PICTURE X.
02 UNIT-CONTROL-WORD PICTURE 9(5).
02 OPTIONAL PICTURE X(66).
```

Format of Required Label on Blank Reels

```
01 AVAILABLE-REEL USAGE IS DISPLAY.
02 LABEL-IDENTIFIER PICTURE X(6).
88 BLANK-TAPE VALUE IS '1BLANK'.
02 FILLER PICTURE X.
02 REEL-SERIAL-NUMBER PICTURE X(5).
02 OPTIONAL PICTURE X(72).
```

IBM COBOL Character Set

The complete IBM COBOL character set consists of the following 48 characters:

0-9	Numbers
A-Z	Letters of the alphabet
	Blank or space
+	Plus sign
−	Minus sign or hyphen
*	Asterisk
/	Stroke (virgule or slash)
=	Equal sign
$	Dollar sign
,	Comma
.	Period or decimal point
"	Quotation mark
(	Left parenthesis
)	Right parenthesis

Collating Sequence

The two permissible collating sequences, in order with the lowest values at the top of the columns, are:

7090/94	COMMERCIAL
0 through 9	blank or space
=	.
'	) or □
	+ or &
+	$
A through I	*
+	−
0	/
.	,
)	(or %
−	= or #
J through R	' or @
−	+
0	0
$	A through I
*	−
blank or space	0
/	J through R
S through Z	≠
≠	S through Z
,	0 through 9
(	

The commercial collating sequence is used by the compiler unless the 7090/7094 collating sequence is specified by placing BINSEQ on the SIBCBC control card.

MAP BCD Character Code and IBM Punched Card Code

The MAP BCD character code is shown in octal form in the following table with the corresponding IBM punched card code.

CHARACTER	BCD CODE (OCTAL)	CARD CODE
(blank)	60	(blank)
0	00	0
1	01	1
2	02	2
3	03	3
4	04	4
5	05	5
6	06	6
7	07	7
8	10	8
9	11	9
A	21	12-1
B	22	12-2
C	23	12-3
D	24	12-4
E	25	12-5
F	26	12-6
G	27	12-7
H	30	12-8
I	31	12-9
J	41	11-1
K	42	11-2
L	43	11-3
M	44	11-4
N	45	11-5
O	46	11-6
P	47	11-7
Q	50	11-8
R	51	11-9
S	62	0-2
T	63	0-3
U	64	0-4
V	65	0-5
W	66	0-6
X	67	0-7
Y	70	0-8
Z	71	0-9
+ (plus)	20	12
− (minus)	40	11
/ (slash)	61	0-1
'(apostrophe)	14	8-4
. (period)	33	12-8-3
) (right parenthesis)	34	12-8-4
$ (dollar sign)	53	11-8-3
* (asterisk)	54	11-8-4
, (comma)	73	0-8-3
((left parenthesis)	74	0-8-4

Alphanumeric Characters Corresponding to Digits with a Sign Overpunch

RIGHTMOST CHARACTER AND SIGN	ALPHANUMERIC CHARACTER
+0	
+1	A
+2	B
+3	C
+4	D
+5	E
+6	F
+7	G
+8	H
+9	I
−0	−
−1	J
−2	K
−3	L
−4	M
−5	N
−6	O
−7	P
−8	Q
−9	R

NOTE: All NUMERIC DISPLAY data-items and negative NUMERIC COMPUTATIONAL data-items are displayed with the sign indicated as an overpunch in the rightmost character position.

Complete List of IBM 7090/7094 COBOL Words

The words listed in this section constitute the complete IBM 7090/7094 COBOL vocabulary. These words are pre-empted and may not be used in a COBOL program except as specified in the manual.

Many words are given in both singular and plural form. This is for the convenience of the source programmer. The compiler recognizes and accepts either form.

ACCEPT	AND
ADD	APPLY
AFTER	ARE
ALL	ASSIGN
ALPHABETIC	AT
ALPHANUMERIC	AUTHOR
ALTER	BCD
AN	BEFORE

BEGINNING
BINARY
BLANK
BLOCK
BY

CALL
CARD-PUNCH
CARD-READER
CHARACTER
CHECK*
CHARACTERS
CHECKPOINT-UNIT
CHECK-SUM
CLASS
CLOSE
COBOL
COLLATE-COMMERCIAL*
COMPUTATIONAL
COMPUTATIONAL-1
COMPUTATIONAL-2
COMPUTE
CONFIGURATION
CONSTANT
CONTAINS
CONTROL
CORRESPONDING
COUNT

DATA
DATE-COMPILED
DATE-WRITTEN
DECLARATIVES
DENSITY
DEPENDING
DIGIT
DIGITS
DISPLAY
DIVIDE
DIVISION
DOLLAR*

ELSE
END
ENDING*
ENTER
ENTRY
ENVIRONMENT
EQUAL
EQUALS
ERROR
EVERY
EXAMINE
EXCEEDS
EXIT
EXPONENTIATED*

FD
FILE
FILES
FILE-CONTROL
FILE-IDENTIFICATION
FILE-SERIAL-NUMBER
FILLER
FIRST
FLOAT*
FOR
FROM

GIVING
GO
GREATER

HIGH
HIGH-VALUE
HIGH-VALUES
HYPERTAPE

IBM-7090
IBM-7094
IBM7090 *
IBM7094 *
ID
IDENTIFICATION
IF
IN
INPUT
INPUT-OUTPUT
INSTALLATION
INTO
I-O-CONTROL
IS

KEY

LABEL
LEADING
LEAVING *
LEFT
LESS
LINKAGE-MODE
LOCATION
LOCK
LOW
LOW-VALUE
LOW-VALUES

MINUS *
MODE
MOVE
MULTIPLE
MULTIPLY

NEGATIVE
NEXT
NO
NONE
NOT
NOTE
NUMERIC

OBJECT-COMPUTER
OCCURS
OF
OFF
OMITTED
ON
OPEN
OPTIONAL *
OR
OTHERWISE
OUTPUT

PERFORM
PICTURE
PLACE
PLACES
PLUS *
POINT
POSITIVE
PRINTER
PROCEDURE
PROCEED
PROGRAM-ID
PROTECT *
PROVIDE

QUOTE
QUOTES

READ
RECEIVE
RECORD
RECORDING
RECORDS
REDEFINES
REEL
REEL-SEQUENCE-NUMBER
REMARKS
RENAMING
REPLACING
RERUN
RETENTION-PERIOD
RETURN
RETURNING
REWIND
RIGHT
ROUNDED
RUN

SECTION
SECURITY
SELECT
SENTENCE
SEQUENCE-CHECK
SIGN
SIGNED
SIZE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAME
STANDARD
STATUS
STOP
SUBTRACT
SYNCHRONIZED
SYSIN1
SYSOU1

TALLY
TALLYING
TAPE-UNIT
TAPE-UNITS
THAN
THEN
THROUGH } equivalent
THRU }
TIME
TIMES
TO

UNEQUAL
UNTIL
UPPER-BOUND*
UPPER-BOUNDS*
UPON
USAGE
USE
USING

VALUE
VARYING
VIA

WHEN
WITH
WITHOUT
WORKING-STORAGE
WRITE

ZERO
ZEROES
ZEROS

---

*This word is not described in this publication but, nevertheless, is preempted and should not be used in a 7090/7094 COBOL program.

Glossary

The following list contains definitions of terms as used in this publication.

ALPHABETIC DATA
A combination of the space and the 26 characters of the alphabet.

ALPHANUMERIC DATA
A combination of the characters in the COBOL character set shown in "Appendix B."

BCD (BINARY CODED DECIMAL)
A code composed of 0's and 1's used for representing alphanumeric data in core storage. Each character in the COBOL character set can be coded as a combination of six 0's and 1's. The octal representation of the BCD character is given in "Appendix B." For further information, see "Computer Data and Instructions," in the publication *IBM 7040 and 7044 Data Processing Systems, Student Text*, Form C22-6732.

BINARY
A code in which numbers are represented to the base 2. Numbers used for computations are stored in the computer in binary form. One binary bit is used for each digit of a binary number and one bit is used for the sign. In the computer, a multiple of six bits is always used with the sign as the leftmost bit. Thus, the number +13 is coded as 001101 and the number −13 is coded as 101101. Numeric computational items (binary numbers) that are synchronized right are stored in computer words by themselves with the sign bit in the leftmost bit of the computer word. For example, the number −13 is stored as 10000000000000000000000000000001101. For further information see "Computer Data and Instructions," in the publication *IBM 7040 and 7044 Data Processing Systems, Student Text*, Form C22-6732.

BLOCK
A physical record.

BLOCKING FACTOR
Number of logical records in a physical record.

BLOCK SEQUENCE NUMBER
The number of the block in each reel of the file; that is, the first block on the reel, the second block on the reel, etc. This number can be put in a block sequence word appended to each block on a file recorded in binary mode.

BOOLEAN OPERATORS
The logical operators AND, OR, and NOT.

BUFFER
An area of core storage reserved by the input-output system for storage of blocks of logical records. A buffer area for input files is used to store logical records after the input-output system has read them from tape, but before they are requested in the program. A buffer area for output files holds processed logical records until a block is complete and the records can be written on magnetic tape. For further information, see the publication *IBM 7090/7094 IBSYS Operating System, Input/Output Control System*, Form C28-6345.

BYTE (7090/7094)
Six binary bits.

CHECKPOINT
A reference point taken during execution of a program. The tape positions, the status of machine registers and switches, and the contents of core storage are recorded for the purpose of later restarting the program.

CHECKSUM
Used to check whether data has been transmitted accurately. It is formed by computing the logical sum of the data in the block, excluding the block sequence word, and logically adding the left and right halves of the word. This computation is performed for input files and checked for output files by the input-output system. The checksum is recorded in the block sequence word that may be appended to the end of each block on a file recorded in binary mode.

COBOL Words
Words that are pre-empted and may be used only as specified in this publication. A complete list of these words is given in "Appendix B."

COMPILE TIME
The time when the user's program is compiled.

COMPLEX NUMBER
A number of the form x + iy where $i = \sqrt{-1}$.

COMPUTER CHARACTER (7090/7094)
One sixth of a computer word. Each computer character is composed of six binary bits, each of which can be *on* (coded as 1) or *off* (coded as 0).

One BCD character can be stored in one computer character; six BCD characters can be stored in one computer word. A binary number that can be converted to 10 decimal digits can be stored in one computer word.

COMPUTER WORD
The basic unit of logic in 7090/7094 computers. Each computer word is composed of 36 binary bits. All computations are performed with computer words.

CONDITION NAME
Name assigned to one value in the range of values a conditional variable may assume. Condition-names are identified by the special level-number 88.

CONDITIONAL EXPRESSIONS
An expression that can be either true or false. For example, A IS GREATER THAN B is a conditional expression since it is true or false depending on the values of A and B when the expression is evaluated.

CONDITIONAL STATEMENT
A statement in the PROCEDURE DIVISION containing a conditional expression to be evaluated.

CONNECTIVES
The words, AND, OR, THEN, and/or a comma used to form compound conditional statements. The words OF and IN used to denote qualification.

CONTROL SECTION
An area of coding, identified by an external name, that is accessible to other segments of a program. Control sections can be replaced with coding in other program segments or can be deleted. Control section names given in a debug request (*name* on the $IBDBC control card) permit the deletion of the debug request at load time.

CORE-STORAGE DUMP
The contents of core storage written on an external unit.

DATA-NAME
Name assigned to identify an area of core storage that is filled with data when the program is executed. The value of the data stored in an area identified by a data-name may vary during the program.

DEFINING NAME
The name following the level-number in the data-item description or the level indicator FD in the file description. Defining names must not be subscripted or qualified.

DENSITY
The number of characters that can be written on a given area of tape; the more characters that can be written, the higher the density. Tapes may be recorded in either HIGH or LOW density.

DIFFERING LENGTH RECORDS
Different record types; that is, more than one record-name in the DATA RECORDS clause of the file description entry.

DUMP
See core-storage dump.

ELEMENTARY ITEM
A data-item that contains no subdivisions.

ENTRY KEYS
Switches on the console of the computer that can be set on or off by the machine operator. They may be tested with a switch-status test in a COBOL program.

EXTERNAL DECIMAL
An elementary data-item described as NUMERIC DISPLAY; that is, a BCD number.

FIGURATIVE CONSTANT
Value with a preassigned fixed data-name. Figurative constants are: ZERO[E][S], SPACE[S], HIGH-VALUE[S], LOW-VALUE[S], QUOTE[S], ALL. Rules for figurative constants are given in "Appendix A."

FILE
A collection of related logical records. A file may contain one or more types of logical records.

FILE CONTROL BLOCK
An area of core storage containing the information about a file that is provided in the file description entry and the FILE-CONTROL paragraph.

FIXED-LENGTH RECORDS
Logical records whose length is specified in the DATA DIVISION, not determined when the program is executed.

FIXED-POINT NUMBER
A number whose USAGE is COMPUTATIONAL or DISPLAY (not COMPUTATIONAL-1 or COMPUTATIONAL-2).

FLOATING-POINT NUMBER
A number of the external form $a * 10^b$ used for computations (described as USAGE IS COMPUTATIONAL-1 or COMPUTATIONAL-2) where a and b are to the base 10. Internally, the numbers are stored in the form $c * 2^d$ where c and d are binary numbers.

GROUP ITEM
A data-item that contains subdivisions.

HEADER LABEL
A label record that is at the beginning of each reel of tape for a file. The format of the IBM standard 84-character header label is given in "Appendix B."

IBJOB PROCESSOR
A major subsystem of the 7090/7094 IBSYS Operating System. The COBOL compiler is a component of the IBJOB processor.

IMPERATIVE STATEMENT
A statement in the PROCEDURE DIVISION consisting of a COBOL verb and its operands. Imperative statements do not contain any conditional expressions to be evaluated; they are commands.

INDEPENDENT ITEM
A data-item described in the WORKING-STORAGE or CONSTANT SECTION that is not related to other items in the section. Independent items are identified by the special level-number 77.

INSERTION CHARACTERS
Editing characters to be placed in a number that is moved to a data-item described with the report form of a PICTURE clause. Insertion characters are: $ − + , . CR DB 0 B.

INSERTION POINT
The paragraph-name or section-name (*location* on the $IBDBC control card) in the program where debug statements are to be executed. Although debug requests are written in a debugging package placed at the end of the program, they are executed as if they were placed in the program following the insertion point.

INTERNAL DECIMAL
An elementary data-item described as NUMERIC COMPUTATIONAL; that is, a binary number.

IOCS (INPUT/OUTPUT CONTROL SYSTEM)
An object-time subroutine that controls data transmittal between storage and external storage devices.

KEY WORDS
See COBOL Words.

LABEL
A record that identifies a file. The standard label format is given in "Appendix B." For further information, see the publication *IBM 7090/7094 IBSYS Operating System, Input/Output Control System*, Form C28-6345.

LITERAL
A word or number that defines itself. The value of the literal is a constant and does not vary in the program. The rules for using literals are given in "Appendix A."

LOGICAL CONNECTIVES
The words AND and OR. For example, A AND B means "both A and B," A OR B means "either A or B."

LOGICAL RECORD
An organized grouping of one or more data-items. The organization is shown by level-numbers in the data-item description entries.

LOOP
A series of statements that are to be executed repeatedly, as under the control of a PERFORM statement.

NUMERIC DATA
Composed of a combination of the decimal point, the plus or minus signs, and the numbers 0 through 9.

OBJECT PROGRAM
A program translated into machine language. This is output from a compilation. It can be loaded into core storage and executed.

OBJECT TIME
The time when the user's program is executed.

OCTAL NUMBER
A number represented to the base 8. Octal numbers are a convenient form for writing binary numbers. Each group of three binary digits can be represented as one octal digit as follows:

BINARY NUMBER	OCTAL EQUIVALENT
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

OPTIONAL WORDS
Words used to improve the readability of the COBOL language. These are upper case, not underlined, words in the formats. Omission of these words is permissible, but misspelling or replacing them by another word constitutes an error.

PARAGRAPH
A group of sentences that conveys one idea or procedure. Paragraphs are used in the PROCEDURE DIVISION and must be given paragraph-names.

PHYSICAL RECORD
Data stored between two interrecord gaps on an external device. A physical record contains one or more logical records.

PROCEDURE-NAME
Paragraph-name or section-name.

PROCEDURE STATEMENT
A conditional or imperative statement.

QUALIFIER
A name preceded by IN or OF that is used to make a subordinate name unique. The rules for qualification are given in "Appendix A."

RECORD
See logical record and physical record.

RECORDING MODE
The code, BCD or binary, in which information is recorded when it is stored on magnetic tape.

REPLACEMENT CHARACTERS
Editing characters that replace nonsignificant zeros in a number that is moved to a data-item described with the report form of a PICTURE clause. Replacement characters are the floating $, −, and + signs.

RERUN
See Restart.

RESERVED WORDS
Words that have special meaning to the compiler and can be used only as specified. See COBOL Words.

RESTART
To resume execution of a program from a reference point established by taking a checkpoint. For further information, see the publication *IBM 7090/7094 IBSYS Operating System, Operator's Guide*, Form C28-6355.

SAME LENGTH RECORDS
Records of the same type; that is, one record-name in the DATA RECORDS clause.

SCALING
1. Automatic process of effectively handling decimal places in arithmetic computations.
2. Indicating the position of an assumed decimal point for a data-item when the point is not within the item. This is done with a P in the PICTURE clause.

SCALING FACTOR
Number of decimal places in an item.

SCIENTIFIC DECIMAL
A report item of the form $a * 10^b$ where a and b are numbers to the base 10. Scientific decimal items are ALPHANUMERIC DISPLAY.

SECTION
A group of paragraphs. Sections must be given section-names.

SENTENCE
One or more procedure statements (possibly separated by a connective) terminated by a period.

SYSTEM UNIT
An input-output device that is named according to its function under the 7090/7094 IBSYS Operating System.

TALLY
A preassigned data-name for a special register (SIZE IS 5 COMPUTATIONAL DIGITS SYNCHRONIZED RIGHT) used to hold intermediate results during a program.

TRAILER LABEL
A label record that is the last logical record on each reel of tape for a file. The format for the standard 84-character trailer label is given in "Appendix B."

TRUNCATION
1. The elimination of leading and/or trailing digits from a number (aligned by a decimal point) that is too large to fit in the space reserved for it.
2. The elimination of trailing characters from a word that is too large to fit in the space reserved for it.

UNPACKING
Putting a data-item in a computer word by itself.

VARIABLE LENGTH
Organization containing an OCCURS DEPENDING ON clause.

VARIABLE-LENGTH RECORDS
Records whose size is not determined until the program is executed; that is, those containing an OCCURS DEPENDING ON clause in the data-item description entry.

WORD
See computer word.

Index

Printed in U.S.A.

C28-6391-0

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601