File No. S360-24
Form C28-6516-2

Systems Reference Library

IBM Operating System/360

COBOL Language

COBOL (Common Business Oriented Language) is similar to English. It was developed by the Conference of Data Systems Languages (CODASYL). COBOL provides a convenient method of coding programs to handle commercial data processing problems.

This publication describes COBOL as implemented for the IBM Operating System/360.

Two COBOL compilers are implemented for Operating System/360, called COBOL E and COBOL F. Differences between the features implemented by the two compilers are discussed in the preface. IBM extensions to COBOL are also discussed in the preface.

This publication discusses the four divisions of a COBOL program and describes the following special features of System/360 COBOL:

1. Report Writer Feature
2. Sort Feature
3. Source Program Library Facility
4. Specifications for the Sterling Currency Feature and International Considerations
5. COBOL Debugging Language

Three appendixes are included:

1. A list of definitions of terms in COBOL formats
2. A COBOL word list
3. A sample problem on asynchronous processing

This publication provides the programmer with rules for writing programs in COBOL for System/360. Users unacquainted with COBOL should first familiarize themselves with the publication: COBOL: General Information Manual, Form F28-8053-2.

PREFACE

The features implemented in COBOL E are a subset of the features implemented in COBOL F. Statements common to both COBOL E and COBOL F, when processed by the COBOL E Compiler, produce results equivalent to those compiled by the COBOL F Compiler. The features of COBOL that are not being implemented in COBOL E are marked in the margin of this publication by the words "F Only." These features are the following:

1. Asynchronous Processing
   a. Options 3 and 4 of the APPLY clause
   b. The USE FOR RANDOM PROCESSING sentence
   c. The HOLD statement
   d. The PROCESS statement
2. The CORRESPONDING option of the ADD, SUBTRACT, and MOVE statements.
3. The Report Writer Feature
4. The Sort Feature
5. Implied subjects and relations in compound conditions
6. Nested IF statements

The following features will be available in COBOL E, but with restricted forms:

1. The DEPENDING ON option of the OCCURS clause
2. The Sterling Currency feature
3. The EXHIBIT statement

The following features will not be available with the initial release of COBOL E:

1. The Extended Source Program Library Facility
2. Random Access
   a. The ORGANIZATION clause
   b. The SYMBOLIC KEY clause
   c. The ACTUAL KEY clause
   d. The clause ASSIGN TO DIRECT-ACCESS
   e. The clause ACCESS IS RANDOM
   f. Option 1 of the APPLY clause
   g. The REWRITE statement
   h. The OPEN statement with the I-O option
   i. The READ and WRITE statements with the INVALID KEY option

However, disk storage devices may be used with standard sequential files (i.e., files assigned as UTILITY).

The following features are IBM extensions to COBOL for Operating System/360 and are marked in the margins of this publication by the word "Ext."

1. The ORGANIZATION clause
2. Internal and external floating-point items and floating-point literals
3. The WITHOUT COUNT CONTROL option of the RECORD CONTAINS clause
4. The interpretation of data-name in the LABEL RECORDS clause
5. The Linkage Section of the Data Division
6. Options 1 and 2 of the USE sentence
7. The REWRITE statement
8. The TRANSFORM statement
9. The Debugging Language
10. Sterling Currency Feature
11. The ACCESS, SYMBOLIC KEY, and ACTUAL KEY clauses have been made part of the Environment Division.

MAJOR REVISION (DECEMBER 1964)

This publication is a major revision of Form C28-6516-1, which is now obsolete.

This publication was prepared for production using an IBM computer to update the text and to control the page and line format. Page impressions for photo-offset printing were obtained from an IBM 1403 Printer with a 120-character chain containing upper and lower case letters, special characters, and numerals.

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

A form for readers' comments appears at the back of this publication. It may be mailed directly to IBM. Address any additional comments concerning this publication to the IBM Corporation, Programming Systems Publications, Department D58, PO Box 390, Poughkeepsie, N.Y. 12602.

ACKNOWLEDGEMENT

The following extract from Government Printing Office Form Number 1962-0668996 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 organizations participating in the original development were:

Air Material 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
Royal McBee 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 assume that a number of improvements and additions will be made to COBOL. Every effort will be made to insure that the improvements and corrections will be made in an orderly fashion, with due recognition of existing users' investments in programming. However, this protection can be positively assured only by individual implementors.

"Procedures have been established for the maintenance of COBOL. Inquiries concerning the procedures and the 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,* 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

---

* Trademark of Sperry Rand Corporation

CONTENTS

This section defines the COBOL character set and describes the formation of COBOL words. It also includes special topics such as punctuation, name qualification, and rules for writing COBOL source programs on a program sheet.

CHARACTER SET

The complete COBOL character set consists of the following 51 characters:

Digits 0 through 9
Letters A through Z
Special characters:

Blank	or space
+	Plus sign
-	Minus sign or hyphen
*	Check protection symbol, asterisk
/	Slash
=	Equal sign
>	Inequality sign (greater than)
<	Inequality sign (less than)
$	Dollar sign
,	Comma
.	Period or decimal point
'	Quotation mark
(	Left parenthesis
)	Right parenthesis
;	Semicolon

Of the previous set, the following characters are used for words:

0 through 9
A through Z
- (hyphen)

The following characters are used for punctuation:

'	Quotation mark
(	Left parenthesis
)	Right parenthesis
,	Comma
.	Period
;	Semicolon

The following characters are used in arithmetic expressions:

+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

The following characters are used in relation tests:

```
>
<
=
```

All of the preceding characters are contained in the COBOL character set. In addition, the programmer can use, as characters in non-numeric literals, any characters (except the quotation mark) included in the IBM Extended BCD Interchange Character (EBCDIC) set; however, such characters may be unacceptable to COBOL for other computers.

PUNCTUATION

The following general rules of punctuation apply in writing COBOL source programs:

1. When any punctuation mark is indicated in a format in this publication, it is required.

2. A comma may be used optionally as a separator between successive operands of a statement, or to separate a series of clauses. A period or comma, when used, must not be preceded by a space, but must be followed by a space.

3. A left parenthesis must not be followed immediately by a space; a right parenthesis must not be preceded immediately by a space.

4. At least one space must appear between two successive words and/or parenthetical expressions and/or literals. Two or more successive spaces are treated as a single space, except in non-numeric literals.

5. When an arithmetic operator or an equal sign is used, it must be preceded by a space and followed by another space.

6. When the period or comma, or arithmetic operator characters are used in the PICTURE clause as editing characters, they are governed by rules for report items only.

7. A semicolon may be used optionally to separate a series of statements or clauses. A semicolon, when used, must not be preceded by a space, but must be followed by a space.

WORD FORMATION

A word is composed of a combination of not more than 30 characters, chosen from the following set of 37 characters:

```
0 through 9 (digits)
A through Z (letters)
- (hyphen)
```

A word must not begin or end with a hyphen. A word is ended by a space, or by proper punctuation. Consecutive embedded hyphens are permitted. All words in COBOL are either reserved words, which have preassigned meanings in COBOL, or programmer-supplied names. Each type of name is discussed in the section of this publication in which it is first mentioned.

QUALIFICATION OF NAMES

Every name used in a COBOL source program must be unique within the source program, either because no other name has the identical spelling, or because the name exists within a hierarchy of names (so that the name can be made unique by mentioning one or more of the higher levels of the hierarchy). The higher levels are called qualifiers when used in this way, and the process is called qualification.

The following rules apply to the qualification of names:

1. The word OF or IN must precede each qualifying name, and the names must appear in ascending order of hierarchy.

2. A qualifier must be of a higher level and within the same hierarchy as the name it is qualifying.

3. The same name must not appear at two levels in a hierarchy in such a manner that it would appear to qualify itself.

4. The highest level qualifier must be unique. Each qualifying name must be unique at its own level within the hierarchy of the immediately higher qualifier.

5. Qualification when not needed is permitted.

6. Qualifiers must not be subscripted, although the entire qualified name may be subscripted.

COBOL PROGRAM SHEET

The purpose of the program sheet is to provide a standard way of writing COBOL source programs.

The Identification, Environment, Data, and Procedure Divisions which constitute a COBOL source program are written in the stated order. This program sheet, despite its necessary restrictions, is of a relatively free form. The programmer should note, however, that the rules for using it are precise and must be followed exactly. These rules take precedence over any other rules, with respect to spacing.

Conceptually, one blank is assumed to be appended after column 72 on every line of a program sheet, except where a non-numeric literal spans more than one line.

SEQUENCE NUMBER: (COLUMNS 1-6)

The sequence number must consist only of digits; letters and special characters should not be used. The sequence number has no effect on the source program and need not be written. If the programmer supplies sequence numbers in each program card, the compiler will check the source program cards and will indicate any errors in their sequence. If these columns are blank, no sequence error will be indicated.

SOURCE PROGRAM STATEMENTS: (COLUMNS 8-72)

These columns are used for writing the COBOL source program.

PROGRAM IDENTIFICATION CODE: (COLUMNS 73-80)

These columns can be used to identify the program. Any character from the COBOL character set may be used, including the blank. The program identification code has no effect on the object program or the compiler.

MARGIN RESTRICTIONS

There are two margins on the COBOL program sheet; Margin A (columns 8-11), and Margin B (columns 12-72).

The names of divisions must begin a Margin A. The division-name must be followed by a space, the word DIVISION, and a period. This entry must appear on a line by itself.

A section-name must begin in Margin A, followed by a space, the word SECTION, and then a period. A paragraph-name must also begin in Margin A and must be followed immediately by a period and a space. Statements may start on the same line in Margin B. Succeeding lines of the paragraph must be written in Margin B.

The level indicators (FD, SD, SA, and RD) of the File, Sort, Saved Area, and Report Description entries in the Data Division, must begin in Margin A. Names and clauses within these entries must not begin before column 12. The level numbers (01-49, 77, 88) of data description entries may begin in Margin A; however, the rest of the entry (data-names and/or clauses) must not begin before column 12.

CONTINUATION OF NON-NUMERIC LITERALS

When a non-numeric literal is of a length such that it cannot be contained on one line of a coding sheet, the following rules apply:

1. On every line containing a portion of a literal to be continued, the portion of the literal that is to be continued must not be terminated with a quotation mark.

2. On every line containing a portion of a literal being continued, the portion of the literal being continued must be immediately preceded by a quotation mark. This quotation mark may appear anywhere in Margin B and may not be preceded by anything but spaces.

3. If compatibility with previous COBOL compilers is desired, a hyphen must be punched in column 7 of each line in which the literal is being continued.

FORMAT NOTATION

Throughout this publication, basic formats are prescribed for various elements of COBOL. These generalized descriptions are intended to guide the programmer in writing his own statements. They are presented in a uniform system of notation, explained in the following paragraphs. This notation is useful in describing COBOL, although it is not part of COBOL.

1. All words printed entirely in capital letters are reserved words. These are words which have preassigned meanings in the COBOL language. In all formats, words in capital letters represent an actual occurrence of those words.

2. All underlined reserved words are required unless the portion of the format containing them is itself optional. These are key words. If any such word is missing or is incorrectly spelled, it is considered an error in the program. Reserved words not underlined may be included or omitted at the option of the programmer. These words are used only for the sake of readability. These words are called optional words.

3. All punctuation and special characters (except those symbols cited in the following paragraphs) represent the actual occurrence of those characters. Punctuation is essential where it is shown. Additional punctuation can be inserted, according to the rules for punctuation specified in this publication.

4. Lower-case words in formats represent information that must be supplied by the programmer. All lower-case words that appear in a format are defined in the accompanying text or in Appendix A.

5. In order to facilitate references to them in text, some lower-case words are followed by a hyphen and a digit or letter. This modification does not change the syntactical definition of the word.

6. Certain hyphenated words in the formats consist of capitalized portions followed by lower-case portions. These designate clauses or statements that are described in other formats, in appropriate sections of the text.

7. Square brackets ([ ]) are used to indicate that the enclosed item may be used or omitted, depending on the requirements of the particular program. When two or more items are stacked within brackets, one or none of them may occur.

8. Braces ({ }) enclosing vertically stacked items indicate that one of the enclosed items is obligatory.

9. The ellipsis (...) indicates that the immediately preceding unit may occur once, or any number of times in succession. A unit means either a single lower-case word, or a group of lower-case words and one or more reserved words enclosed in brackets or braces. If a term is enclosed in brackets or braces, the entire unit of which it is a part must be repeated when repetition is specified.

10. Comments, restrictions, and clarifications on the use and meaning of every format are contained in the appropriate portions of the text.

System/360 COBOL provides the user with two different programming techniques. One of these is the conventional programming technique now used by most programmers (called synchronous processing). The other is a new technique that provides for COBOL programs with multiprogramming capabilities (called asynchronous processing).

SYNCHRONOUS PROCESSING

Instructions in programs processed synchronously are executed in a sequential manner. Synchronous processing is normal for applications where the records are processed in the order in which they are read. These records may be accessed either sequentially or randomly. (See "Access Methods" under Input/Output Processing, in the text below).

ASYNCHRONOUS PROCESSING

F Only

Asynchronous processing is designed primarily to allow the COBOL user to take full advantage of available direct-access devices, and of the multiprogramming feature of Operating System/360, with a minimum loss of input/output time.

Programs to be asynchronously processed are made up of two parts; the in-line portion, and the out-of-line portion. The in-line portion is processed in the same manner as synchronously processed programs. The primary purpose of an in-line procedure in an asynchronously processed program is to initiate the out-of-line portion of the program and to control the flow of data to be processed by the out-of-line procedures. Each execution of the out-of-line portion of a program initiated by a PROCESS statement is a cycle. The PROCESS statement (described in detail in the Procedure Division) is used to initiate the out-of-line portion. The PROCESS statement may not appear in synchronously processed programs. The out-of-line portion is written in a USE FOR RANDOM PROCESSING Section in the Declaratives portion of the Procedure Division.

The in-line portion runs continuously, independent of the out-of-line cycles. During the in-line processing the PROCESS statement is executed many times. At each execution, a new out-of-line cycle is initiated. Once initiated, each out-of-line cycle operates independently of the in-line portion and any other out-of-line cycle. This allows the user to overlap random accessing of a data record with the processing of other records.

Usually, only the random access method is used in the out-of-line procedures. However, there is a facility enabling the user to force sequential processing of data in an out-of-line procedure. The HOLD statement (described in detail in the Procedure Division) is used for this purpose. Subsequent statements may use either sequential or random access, and are processed synchronously.

Because of the characteristics of the out-of-line procedures, some COBOL statements are not allowed. These restrictions are indicated in

the appropriate statement descriptions appearing in the Procedure Division section of this publication.

Although the in-line portion and the out-of-line cycles effectively operate continously and independently, they actually operate interleaved. The in-line portion or an out-of-line cycle retains control until it has to wait for the completion of an input/output operation, or the completion of another cycle. Control is then passed to another cycle (or the in-line portion) that is not in a waiting status. A cycle that is waiting is resumed only after the condition for which it is waiting is satisfied.

F Only ASYNCHRONOUSLY PROCESSED FILE AREAS

Because of multiple cycles for an out-of-line portion of a program, a method providing for the concurrent existence of multiple data records associated with each cycle is provided.

A saved area (defined by an SA entry in the Data Division) provides an area for transferring data from the in-line portion to an out-of-line cycle that may serve as a working-storage area for the out-of-line cycle. One representation of the defined saved area is available for each cycle allowed in the program (see Option 4 of the APPLY clause in the Environment Division).

The records of a file to which an out-of-line procedure refers are processed in separate areas for each cycle. Records in a file are defined by record description entries in the Data Division.

INPUT/OUTPUT PROCESSING

System/360 COBOL supports various data organizations, record formats, and access methods. The facilities available to the COBOL user are specified in this section.

In this publication, the term "file" can be considered equivalent to the term "data set" used in other Operating System/360 publications. In addition, in this publication, the term IOCS (Input/Output Control System) can be considered equivalent to the term "Data Management Routines" used in other Operating System/360 publications. Further information on data sets is contained in the publication: IBM Operating System/360: Data Management.

Data Organization

System/360 COBOL provides three types of data organization: standard sequential, indexed, and relative.

The number and type of control fields used to locate logical records in a file differ, depending on which of these three types of data organization is used. Consequently, each type of data organization is incompatible with the other two. For example, records created on a standard sequential file cannot also be read as an indexed file.

Standard Sequential Data Organization

When standard sequential data organization is used, the logical records in a file are positioned sequentially in the order in which they are created, and are read sequentially in the order in which they were created (or in sequentially reversed order if the REVERSED option of the OPEN statement is written). This type of data organization is used for tape files but it is device-independent; standard sequential files may be assigned to utility, direct-access, or unit-record devices.

Indexed Data Organization

When indexed data organization is used, the position of each logical record in a file is determined by indexes maintained by the system and created with the file. The indexes are based on symbolic keys provided by the user. Indexed files must be assigned to direct-access devices.

Relative Data Organization

When relative data organization is used, the positioning of the logical records in a file is determined by an actual key supplied by the user. Symbolic keys are used in relative data organization to identify records, not to position them. Actual keys specify the track, relative to the first track for a file, on which each record is to be placed. On each track, records are positioned in the order in which they are written. Relative files must be assigned to direct-access devices.

Access Methods

There are two access methods provided by System/360 COBOL: sequential access and random access.

SEQUENTIAL ACCESS: This is the method of reading and writing records of a file in a serial manner; the order of references is implicitly determined by the position of a record in the file.

Sequential access may be used for files organized in any of the three data organizations previously described.

RANDOM ACCESS: This is the method of reading and writing records of a file in an arbitrary, non-sequential manner; the control of successive references to the files being determined by specifically defined keys supplied by the user.

Random access may be used anywhere with either type of COBOL processing, but may only be used for files having indexed or relative organization.

Keys

When accessing indexed or relative files randomly, the user must provide information to identify the specific record desired. For both

of the organizations, the user must provide a symbolic key for the desired record. In addition, for a relative file, the user must also provide the relative track number (actual key) on which the record is located. The symbolic key is normally a unique value which distinguishes a record from all other records in the file (e.g., a stock-number in an inventory file or an employee's name or man-number in a payroll file). In an indexed organization file the symbolic key for each record must be unique. In a relative organization file, however, the user may specify the same symbolic key for more than one record, provided that the records are not on the same track.

The relative track number (actual key) is a value that is between zero and the number of tracks available to the file minus one. For example, if a file has 100 tracks available to it, the relative track numbers will range from 0 to 99 even though the actual tracks are physically noncontiguous or even located on more than one device.

The symbolic and actual keys are used as follows when a file is accessed randomly:

1. For a relative file, IOCS uses the relative track number to determine the actual track address. After locating the track, for a read or rewrite operation, IOCS searches the track for a record which has a "key field" equal to the record identity. When a match is found, the data portion of the record is read or, if a rewrite, replaced by the new record. For a write operation, after locating the actual track, IOCS searches for the last record on the track and writes the new record (with control fields including a key field equal to the symbolic key provided). If the desired record cannot be found on the specified track, IOCS will search the entire file for the record. For a write operation, if a record will not fit on the specified track, IOCS will search the entire file for a free area large enough to hold the new record, except when an Option 1 APPLY clause is written in the Environment Division.

2. For an indexed file, the track on which the record with the given symbolic key is located is determined by using the symbolic key and searching the file's index table. When the track has been determined, a read and rewrite operation is the same as described above. For a write operation, the functions performed are different from those for a relative organization file, in that the record is written (including control fields) between the two records on the track which have symbolic keys lower and higher than the symbolic key for the new record. If the insertion of the new record causes a record to overflow the end of the track, this overflow record is placed into an "overflow area" for the file and the file's indexes are updated to reflect the revised positioning of records. Since the track on which each record in the file is located can be determined from the indexes, only the determined track is searched for the record.

If an indexed or relative file is accessed sequentially, the operation is similar to that for a standard sequential file. That is, IOCS determines where a record is to be found based solely upon the logical sequence in which records were placed in the file previously. For relative files, this logical sequence corresponds exactly to physical sequence; for indexed files, this logical sequence corresponds to the sequence of keys. Therefore, when accessing sequentially, the user does not specify a symbolic key or actual key. An option is provided, however, for the user to request that, with each record read, the symbolic key also be read and made available. Similarly, the actual key (relative track number) may also be optionally obtained when sequentially processing a relative file.

It should be noted that the above discussion applies specifically to files accessed or created by a COBOL program. It is possible in lower

level languages to create indexed or relative files which contain logical records and no keys. In general, such files may not be used by COBOL object programs.

When an indexed or relative organization file is accessed randomly, the symbolic key of the desired record must be moved into the data-name specified by the SYMBOLIC KEY clause and, for a relative organization file, the relative track number must be moved into the data-name specified by the ACTUAL KEY clause before the read, write or rewrite for the record is executed. These values are used by IOCS, to determine where the record is located or where it should be placed. For a randomly accessed file the values of the data-names for the symbolic and actual keys are never automatically modified by IOCS. The user has complete responsibility for insuring that the correct values are in the data-names before reading, writing, or rewriting.

When an indexed or relative file is accessed sequentially, no key specifications are necessary since IOCS determines implicitly where the records are located. If the SYMBOLIC KEY and/or the ACTUAL KEY clause is specified, IOCS will return the symbolic and/or actual key for the record read to the specified data-names after each read statement is executed. In this case, IOCS controls the values of the data-names and any changes the user makes to their contents will not affect the sequence in which records are read or rewritten.

Only the SYMBOLIC KEY clause may be specified for an indexed file; if access is random, the clause is required and, if access is sequential, the clause is optional.

For a relative file, both the SYMBOLIC KEY and the ACTUAL KEY clauses are allowed. If access is sequential, both clauses are optional (either one or both or none may be specified).

Summary Figure

Figure 1 summarizes the clause and statement specifications allowed for each of the three data organizations. In addition, each file-name must be specified in a SELECT clause in the Environment Division and must be defined by an FD entry in the File Section of the Data Division.

ORGANIZATION IS	ASSIGN TO	ACCESS IS	SYMBOLIC KEY	ACTUAL KEY	OPEN	Other Allowable I-O Verbs*	Other Allowable Clauses
Not Specified (Standard Sequential)	UTILITY UNIT-RECORD or DIRECT-ACCESS	Not Specified or SEQUENTIAL	Not Allowed	Not Allowed	INPUT OUTPUT	READ WRITE	APPLY condition-name SAME [RECORD] AREA BLOCK CONTAINS RECORD CONTAINS DATA RECORDS REPORTS ARE ** RESERVE LABEL RECORDS VALUE OF RERUN
INDEXED**	DIRECT-ACCESS	Not Specified or SEQUENTIAL	Optional	Not Allowed	INPUT I-O	READ READ REWRITE	SAME [RECORD] AREA APPLY RECORD PROTECTION APPLY section-name *** RECORD CONTAINS LABEL RECORDS VALUE OF DATA RECORDS
		RANDOM	Required	Not Allowed	INPUT OUTPUT I-O	READ WRITE READ REWRITE WRITE	
RELATIVE**	DIRECT-ACCESS	Not Specified or SEQUENTIAL	Optional	Optional	INPUT	READ	SAME [RECORD] AREA APPLY RESTRICTED SEARCH APPLY RECORD PROTECTION APPLY section-name *** RECORD CONTAINS LABEL RECORDS VALUE OF DATA RECORDS
		RANDOM	Required	Required	INPUT OUTPUT I-O	READ WRITE READ REWRITE WRITE	

* CLOSE is permitted with any type of organization.

** F only.

*** May only refer to files which are ACCESS RANDOM.

Figure 1. Permissible Data Organization Clauses And Statements.

The Identification Division is used to identify a program and to provide other pertinent information concerning the program. The format of the Identification Division is:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. program-name. [sentence...]
[AUTHOR. sentence...]
[INSTALLATION. sentence...]
[DATA-WRITTEN. sentence...]
[DATE-COMPILED. sentence...]
[SECURITY. sentence...]
[REMARKS. sentence...]
```

Program-name must consist of from one to eight letters and/or digits. The first character of program-name must be a letter. Program-name identifies the object program to the Control Program.

IDENTIFICATION and the other COBOL words in the Identification Division must begin in Margin A. If sentences are written, they must begin in Margin B. They may consist of any characters in the EBCDIC set.

GENERAL DESCRIPTION

The function of the Environment Division is to centralize the aspects of the total data processing problem that are dependent upon the physical characteristics of a specific computer. It provides a linkage between the logical concept of files and their records, and the physical aspects of the devices on which files are stored.

The Environment Division is divided into two sections - the Configuration Section and the Input-Output Section.

The Configuration Section, which deals with the over-all specifications of computers, is divided into two paragraphs. They are: the Source-Computer paragraph, which defines the computer on which the COBOL compiler is to be run, and the Object-Computer paragraph, which defines the computer on which the program produced by the COBOL compiler is to be run.

The Input-Output Section deals with the definition of the external media (input/output devices) and information needed to create the most efficient transmission and handling of data between the media and the object program. This section is divided into two paragraphs. They are the File-Control paragraph, which names and associates the files with the external media; and the I-O-Control paragraph, which defines special input-output techniques, including rerun checkpoints.

CONFIGURATION SECTION

The format of the Configuration Section is:

```
CONFIGURATION SECTION.

SOURCE-COMPUTER.  IBM-360[model-number].

OBJECT-COMPUTER.  IBM-360[model-number].
```

This complete section or either of the paragraphs is optional. Model-number is the model number of the system being used, e.g., E30, F50.

INPUT-OUTPUT SECTION

The format of the Input-Output Section is:

```
INPUT-OUTPUT SECTION.
FILE-CONTROL.
     {SELECT file-name ASSIGN-clause
     [RESERVE-clause]
Ext  [ACCESS-clause]
Ext  [ORGANIZATION-clause]
```

```
     [SYMBOLIC KEY-clause]                                          Ext
     [ACTUAL KEY-clause].  }...                                     Ext
I-O-CONTROL.
     [SAME-clause.] ...
     [RERUN-clause.] ...
     [APPLY-clause.] ...
```

The individual optional clauses that compose the File-Control and I-O-Control paragraphs may appear in any order within their respective sentences or paragraphs. They are described in the following text. The Input-Output Section may be omitted if there are no files used in the program.

For a sort-file, the only clause that may be written in the SELECT sentence is the ASSIGN clause. F Only

I-O-CONTROL may be omitted if none of the clauses in the paragraph is written. A period must follow the last clause in each SELECT sentence written in the File-Control paragraph, and must follow each clause written in the I-O-Control paragraph.

FILE-CONTROL PARAGRAPH

SELECT Sentence

The SELECT sentence must begin with the words SELECT file-name and must be given for each file named in the File-Control paragraph.

The name of each file must be unique within a program and must have a File Description (FD or SD) in the Data Division of the source program. Conversely, every file named in an FD or SD entry must be named in a SELECT sentence.

ASSIGN Clause

The format of the ASSIGN clause is:

```
ASSIGN TO {DIRECT-ACCESS [device-number UNIT [S] ]}
          {UTILITY [device-number UNIT [S] ]      }
          {UNIT-RECORD device-number UNIT [S]     }
```

The ASSIGN clause is used to assign a file to a particular device.

DIRECT-ACCESS, UNIT-RECORD, and UTILITY specify device classes. Each file must be assigned to a device class. Files assigned to UTILITY have sequential access only, and data contained on these files is organized in the standard sequential fashion. Files assigned to DIRECT-ACCESS may have standard sequential, indexed, or relative organization. When organization is indexed or relative, access may be either sequential or random.

Device-number is used to specify a particular device type within a device class. For unit-record assignment, device-number is required. Device-numbers must be given when the Procedure Division includes device-dependent statements.

The allowable device-numbers are:

DIRECT-ACCESS
1302, 7320, 2301, 2311 , 2321

UNIT-RECORD
1402, 1442, 1403, 1404, 2201, 1443, 1445

UTILITY
2400,7340,1302, 7320, 2301, 2311, 2321

F Only Note that a sort-file cannot be assigned to device-number 2321.

Ext ACCESS Clause

The format of the ACCESS clause is:

```
[ACCESS IS  {SEQUENTIAL}]
            {RANDOM    }
```

The ACCESS clause indicates the manner in which the records of a file are read or written.

If this clause is not written, ACCESS IS SEQUENTIAL is assumed. If ACCESS IS RANDOM is written, the file must be assigned to a DIRECT-ACCESS device.

Ext ORGANIZATION Clause

The format of the ORGANIZATION clause is:

```
[ORGANIZATION IS{INDEXED }]
                {RELATIVE}
```

This clause may only be written for files assigned to direct-access devices in a SELECT sentence. If ACCESS IS RANDOM is written, this clause must be written.

If the ORGANIZATION clause is omitted, a standard sequential file is assumed.

INDEXED specifies indexed data organization.

RELATIVE specifies relative data organization.

RESERVE Clause

The format of the RESERVE clause is:

```
[RESERVE{NO     } ALTERNATE AREA [S]]
        {integer}
```

This clause specifies the number of buffers reserved for a sequential file in addition to the standard minimum of one required for a file. If

this clause is omitted, integer is assumed equal to 1. If NO is written, no additional buffer will be reserved.

SYMBOLIC KEY Clause Ext

The format of the SYMBOLIC KEY clause is:

[SYMBOLIC KEY IS data-name]

This clause is allowed only when the ORGANIZATION clause is specified, and is required if ACCESS IS RANDOM is specified.

When synchronous processing is used, data-name must not be defined in the file for which it is the symbolic key.

When asynchronous processing is used, data-name must be an item defined in the saved-area associated with the out-of-line procedure that processes the file for which data-name is the symbolic key. F Only

If the SYMBOLIC KEY clause is used for an ACCESS SEQUENTIAL file having an ORGANIZATION clause, the symbolic identity of the record will be placed into data-name whenever a READ statement is executed for the file. Any changes the programmer may make to data-name will not affect the order in which records are read from the file.

If the file is specified as ACCESS RANDOM, the symbolic identity of the desired record to be read or written must be placed in data-name before the READ or WRITE statement for the record is executed. The symbolic identity will be transmitted to IOCS and is used to determine the physical location from which the record is to be read or onto which it is to be written.

Data-name may be any fixed length item less than 256 bytes in length. A discussion of data items is contained in Section 5.

ACTUAL KEY Clause Ext

The format of the ACTUAL KEY clause is:

[ACTUAL KEY IS data-name]

This clause is allowed for a file only when ORGANIZATION IS RELATIVE is specified for it, and is required when ACCESS IS RANDOM is specified for it. This clause must not be specified for a file under any other circumstances.

The functions of this clause are similar to those of the SYMBOLIC KEY clause, except that this clause specifies a data item that will contain the relative track number in a file on which a record is to be found or placed.

When synchronous processing is used, data-name must not be defined in the file for which it is the actual key. When asynchronous processing is used, data-name must be an item defined in the saved area associated with the out-of-line procedure that processes the file for which data-name is the actual key.

Data-name must be defined as a half-word binary data item (i.e., USAGE COMPUTATIONAL and having a PICTURE not greater than 9999).

SAME Clause

The format of the SAME clause is:

```
[SAME [RECORD] AREA FOR file-name-1 file-name-2
      [file-name-3...] .]
```

The SAME clause is used to specify that two or more files are to use the same main storage area for processing.

If the RECORD option is not specified, no more than one of the files named in this clause may be open (by means of an OPEN statement) at any time.

The RECORD option, when specified, allows the user to specify that two files involved in a file update operation are to share the same storage area (IOCS cyclic buffering). The following restrictions apply when the RECORD option is used:

1. Only two files may be specified in the SAME AREA clause.
2. Both files must be open at the same time.
3. One file must be input and one must be output.
4. The processing cycle for the files must be:
   a. read input file-record.
   b. update record.
   c. write updated record on output file.
5. The length of the records cannot be changed in the updating process.
6. No records can be added to or deleted from the file (i.e., every record must be written).
7. The blocking factor (number of logical records per physical record) of the files must be the same for both files.

More than one SAME clause may appear in a COBOL program, but any one file-name may appear in only one SAME clause.

This clause cannot refer to file-names which represent sort-files. (Sort-files are discussed in Section 8.)

RERUN Clause

The format of the RERUN clause is:

```
[RERUN EVERY {END OF UNIT OF file-name}  .]
             {integer CLOCK-UNITS      }
```

This clause may only be written for programs that are synchronously processed. It specifies that checkpoint records are to be written on the standard system checkpoint device.

A checkpoint record is a recording of the status of the computer at a given point in the execution of the object program. It contains all of the information necessary to restart the program from that point.

If the END OF UNIT option is specified, checkpoint records will be written whenever a change of volume occurs for the file named by file-name. File-name must be the name of a standard sequential file. The term volume is defined in Section 5, under "LABEL RECORDS Clause."

If the CLOCK-UNIT option is specified, checkpoint records are written after the expiration of the number of clock units specified by integer. A System/360 COBOL clock unit is one minute.

APPLY Clause

The formats of the APPLY clause are:

Option 1

[APPLY RESTRICTED SEARCH OF integer TRACKS

TO file-name... .]

Option 1 may only refer to files specified as ACCESS IS RANDOM and ORGANIZATION IS RELATIVE. In normal operation, execution of a READ, WRITE, or REWRITE statement for a file causes the entire file to be searched for the specified record when the record cannot be found on the specified relative track. When RESTRICTED SEARCH is written, the search is limited to the specified number of relative tracks. If the desired record cannot be found in the case of a READ or REWRITE, or will not fit on the specified tracks in the case of a WRITE, the INVALID KEY option of the READ, WRITE, or REWRITE Statement will be executed.

The INVALID KEY option will also be executed when the specified relative track is outside of the limits of the file. The programmer must provide his own determination of which condition produced the invalid key condition.

Option 2

[APPLY condition-name TO FORM-OVERFLOW OF file-name.]

This option is used to specify a condition-name which may be used in a condition-name test for form-overflow of a printer to which the file named by file-name is assigned. The condition-name is true if a 'form-overflow' situation exists.

Form-overflow exists when an end-of-page is sensed by an on-line printer.

When this clause is written, unbuffered output is assumed. This is equivalent to writing RESERVE NO ALTERNATE AREAS for an unblocked file assigned to a unit-record printer.

Such a condition-name test may be written in conjunction with a WRITE statement with an ADVANCING option in order to control spacing of printed records. Thus, the following statement could be written (with a programmer-supplied condition-name):

```
IF condition-name WRITE X AFTER
ADVANCING 0 LINES ELSE WRITE X
AFTER ADVANCING 2 LINES
```

Condition-names are discussed in Section 5; condition-name tests are discussed in Section 6.

Option 3 F Only

[APPLY RECORD PROTECTION TO file-name... .]

The files named in this clause must be specified as I-O in an OPEN statement, and must be named in an Option 4 APPLY clause. This clause specifies that an individual record (of the files named) used in one cycle is protected from manipulation by another cycle in an asynchronous program. A cycle that reads a record (by means of a READ statement) must execute a REWRITE statement for the record before another READ statement can be executed for the record. This prevents successive updatings of a record from being made on obsolete or erroneous data.

In order to free a record from being record protected, the following step must be taken for each cycle, even if no alteration to the contents of the record is made: a REWRITE statement for each record that is read must be executed before the execution of any out-of-line HOLD statement.

F Only Option 4

[APPLY section-name TO saved-area-name file-name . . .

[FOR integer CYCLES] .]

Option 4 is used to relate an out-of-line procedure to the data areas it is to process.

Section-name is the name of an out-of-line procedure defined by a USE FOR RANDOM PROCESSING declarative.

Saved-area-name is the name of the saved area that is processed by the out-of-line procedure named by section-name and must be defined by an SA entry in the Data Division.

The file-names are the names of the files that are processed randomly by the out-of-line procedure. All references to the files named in this clause (except for OPEN and CLOSE statements, which must be executed in in-line procedure) must be executed in the out-of-line procedure named by section-name.

The same section-name, saved-area-name, or file-name may not appear in more than one Option 4 clause.

FOR integer CYCLES defines the maximum number of asynchronous processing cycles that are to be active at one time in the out-of-line procedure. It should be noted that the storage requirements of the program will depend upon the number of cycles specified.

If FOR integer CYCLES is omitted, 5 cycles will be assumed by the compiler.

GENERAL DESCRIPTION

The Data Division of a COBOL source program describes the information to be processed by the object program. This information falls into the following categories:

1. Data contained in files, entering or leaving the internal storage of the computer.

2. Data developed internally and placed in intermediate or working storage, and constant data defined by the user.

3. Linkage data descriptions for communication between main program and subprograms.

4. Report format specifications and data to be included in the report.

The Data Division begins with the header DATA DIVISION followed by a period. Each of the sections of the Data Division begins with a fixed section-name, and is followed by the word SECTION and a period, as follows:

```
DATA DIVISION.
FILE SECTION.
    File Description entries
       Record Description entries
    Sort Description entries                                  F Only
       Record Description entries
    Saved Area entries                                        F Only
       Record Description entries
WORKING-STORAGE SECTION.
    Record Description entries
LINKAGE SECTION.                                              Ext
    Record Description entries
REPORT SECTION.        (for Report Writer)                    F Only
    Report Description entries
       Report Group Description entries
       Report Element Description entries
```

The sections must appear in this order. If any section is not required, both it and its section-name may be omitted.

ORGANIZATION OF THE DATA DIVISION

The Data Division is subdivided into sections, according to types of data. Each section consists of entries, rather than sentences and paragraphs. An entry consists of a level indicator, a data-name or file-name, and a series of clauses which may be separated by commas or semicolons. The clauses may be written in any sequence (except the REDEFINES clause). Each entry must terminate with a period and a space.

The File Section describes the content and organization of files and, for COBOL F only, saved-areas and sort-files. Each such entry is followed by related Record Description entries.

The Record Description entries used in conjunction with a File and/or Sort Description entry describe the individual items contained in a data record of a file.

The Working-Storage Section consists solely of Record Description entries. These entries describe the areas of storage where intermediate results are stored at object-program execution time, and constants along with their values.

Ext The Linkage Section is a required part of any COBOL subprogram that contains an ENTRY statement with USING option, and serves as a data-linking mechanism between the main program and the subprogram. It consists only of Record Description entries that provide dummy names for linkage to data in the main program. This is the only Data Division section whose entries do not cause object program data storage areas to be allocated.

F Only The Report Section describes the physical aspects of the report format and the conceptual characteristics of the data. It has three types of entries: the Report Description entry, which specifies the information pertaining to the over-all format of the named report; the Report Group Description entry, which describes the characteristics for a report group; and the Report Element Description entry, which defines the characteristics of each group or elementary item included within a report group.

CONCEPTS OF DATA DESCRIPTION

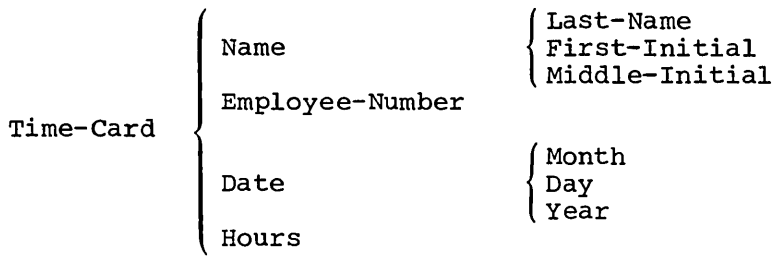
The following material defines the basic terms and concepts used in describing data. Rules which govern the writing of data descriptions appear later in this section.

LEVELS

Level numbers are used to show how data items are related to each other. The most inclusive grouping of data is the file. The level indicator for an input/output file is FD; for a sort file (for COBOL F only) the level indicator is SD.

For purposes of processing, the contents of a file are divided into logical records, with level number 01 specifying a logical record. The object program locations of all logical records are automatically adjusted to the appropriate double-word boundary. Subordinate data items that constitute a logical record are grouped in a hierarchy, and identified with level numbers 02 to 49. Level number 77 identifies a special type of entry in the Linkage Section or the Working-Storage Section. Level number 88 is used to define a condition-name for a related conditional variable. A level number less than 10 may be written as a single digit.

Levels allow specification of subdivisions of a record necessary for referring to data. Once a subdivision is specified, it may be subdivided further to permit more detailed data reference. This may be illustrated by the following weekly time-card record, which is divided into four major items: name, employee-number, date, and hours, with more specific information appearing for name, and date.



Subdivisions of a record, that are not themselves further subdivided, are called elementary items. Data items that contain subdivisions are known as group items. When a Procedure Division statement makes reference to a group item, the reference applies to the area reserved for the entire group. Less inclusive groups are assigned higher level numbers. Level numbers of items within groups need not be consecutive. A group includes all groups and elementary items described under it until a level number less than or equal to the level number of the group is encountered. Separate entries are written in the source program for each level. To illustrate level numbers and group items, the weekly time-card record in the previous example may be described by Data Division entries having the following level numbers and data-names described in Figure 2.

```
01 TIME-CARD
04 NAME
06 LAST-NAME
06 FIRST-INITIAL
06 MIDDLE-INITIAL
04 EMPLOYEE-NUMBER
04 DATE
05 MONTH
05 DAY
05 YEAR
04 HOURS
```

Figure 2. Example of Data Levels.

Only the level number and data-name of each entry have been given in Figure 2.

Level number 77 identifies a second description entry for independent Working-Storage items. The level number 77 cannot appear in the File Section or the Report Section.

Level number 88 designates a condition-name.

CONDITION-NAMES

The general form of a condition-name entry is:

88 condition-name VALUE IS literal.

Condition-names must be formed according to the rules for data-name formation. A level 88 entry must be preceded by either another level 88 entry (in the case of several consecutive condition-names pertaining to an elementary item), or by an elementary item.

Every condition-name pertains to an elementary item in such a way that the condition-name may be qualified by the name of the elementary item and the elementary item's qualifiers. A condition-name is used in the Procedure Division in place of a simple relational condition.

A condition-name may pertain to an elementary item (a conditional variable) requiring subscripts. In this case the condition-name, when written in the Procedure Division, must be subscripted according to the same requirements of the associated elementary item. Subscripting is discussed later in this text.

The literal in a condition-name entry must be consistent with the data type of the conditional variable.

Figure 3 is an example of Data Division entries and a Procedure Division statement that might be written using level 88 and the condition-name-test. (Details on the condition-name-test appear in Section 6 under the subsection "Test-Conditions.")

Data Division Portion:

```
01  TIME-CARD.
02 NAME, PICTURE  X(20).
02 PAY-CODE, PICTURE 9.
    88  MONTHLY, VALUE IS 1.
    88  HOURLY, VALUE IS 2.
    88  SUBCONTRACTOR, VALUE 3.
02 SALARY, PICTURE 9999.
02 RATE-PER-HOUR, REDEFINES SALARY PICTURE 9V999,
   DISPLAY.
02 PER-DIEM, REDEFINES RATE-PER-HOUR PICTURE 99V99,
   DISPLAY.
```

Procedure Division Portion:

```
IF HOURLY COMPUTE GROSS = 40 * RATE-PER-HOUR,
ELSE IF MONTHLY COMPUTE GROSS = SALARY / 4.334,
ELSE IF SUBCONTRACTOR COMPUTE GROSS = 5 * PER-DIEM,
ELSE PERFORM ERROR-PROCESS.
```

Figure 3. Condition-name Example.

DATA-NAMES

Data-names are names assigned by the programmer to identify data items used in a program. They always refer to a kind of data, not to a particular value, and the items they refer to usually assume a number of values during the course of a program.

A data-name must contain at least one alphabetic character. A data-name or the key word FILLER must be the first word following the level number in each Record Description entry, as shown in the following general format:

level-number { data-name / FILLER }

This data-name is the defining name of the entry, and is the means by which references to the associated data area (containing the value of a data item) are made.

If some of the characters in a record are not used in the processing steps of a program, then the data description of these characters need not include a data-name. In this case, FILLER is written in lieu of a data-name after the level number.

If the same data-name is assigned to more than one item in a program, it must be qualified in all references to it in the Procedure Division, Data Division, or Environment Division, except in the REDEFINES clause.

A data-name is qualified by writing either IN or OF after it, followed by the name of one or more groups, or the record or file in which it is contained. A highest level qualifier must, however, be unique.

Any combination of qualifiers that will ensure uniqueness may be used. More qualifiers may be used than are absolutely needed. In Figure 2, if YEAR OF DATE is needed to make YEAR unique, YEAR OF DATE IN TIME-CARD is also permitted.

A data-name cannot be subscripted when it is used as a qualifier. However, the entire qualified data-name may be subscripted.

LITERALS

A literal is a constant that is not identified by a data-name in a program, but is completely defined by its own identity. A literal is either non-numeric (alphabetic or alphanumeric), numeric, or floating-point.

Non-Numeric Literals

A non-numeric literal must be bounded by quotation marks and may consist of any combination of characters in the IBM EBCDIC set, except quotation marks. All spaces enclosed by the quotation marks are included as part of the literal. A non-numeric literal may not exceed 120 characters in length.

The following are examples of non-numeric literals:

```
'EXAMINE CLOCK NUMBER'
'12565'
'PAGE 144 MISSING'
```

Numeric Literals

A numeric literal must contain at least one and not more than 18 digits. A numeric literal may consist of the characters 0 through 9, the plus sign or the minus sign, and the decimal point. It may contain only one sign character and only one decimal point. The sign, if present, must appear as the leftmost character in the numeric literal. If a numeric literal is unsigned, it is assumed to be positive.

A decimal point may appear anywhere within the numeric literal, except as the rightmost character. If a numeric literal does not contain a decimal point, it is considered to be a whole number.

The following are examples of numeric literals:

```
1506798
+12572.6
-256.75
.16
```

Ext Floating-Point Literals

A floating-point literal must have the form:

[±]mantissa E[±]exponent

The plus or minus signs preceding the mantissa and exponent are the only optional characters within the format. The mantissa consists of one to 16 digits with a required leading or embedded decimal point.

Immediately to the right of the mantissa, the exponent is represented by the symbol E, followed by a plus or minus sign (if a sign is given), and one or two digits. The magnitude of the number represented by a floating-point literal must not exceed .72*(10**76). A zero exponent must be written as 0 or 00.

The value of the literal is the product of the mantissa and ten raised to the power given by the exponent. A floating-point literal must appear as a continuous string of characters with no intervening spaces.

The following are examples of floating-point literals:

```
12.3E2
-.34566E+17
+2.56E-6
```

FIGURATIVE CONSTANTS

Figurative constants are a special type of literal. They are values that have been assigned fixed data-names. Figurative constants must not be bounded by quotation marks.

ZERO may be used in many places in a program as a numeric literal. The use of ZERO as a non-numeric literal is permitted. All other figurative constants are considered non-numeric. The singular and plural forms of figurative constants are equivalent and may be used interchangeably.

ZERO ZEROS ZEROES	Represents one or more zeros.
SPACE	Represents one or more blanks or spaces.
HIGH-VALUE HIGH-VALUES	Represents one or more appearances of the highest value in the computer's collating sequence.
LOW-VALUE LOW-VALUES	Represents one or more appearances of the lowest value in the computer's collating sequence.
ALL 'character'	Represents one or more occurrences of the single character bounded by

quotation marks. Character may not be a quotation mark.

QUOTE QUOTES	Represents the character '. Note that the use of the word QUOTE to represent the character ' at object time is not equivalent to the use of the symbol ' to bound a non-numeric literal.

When a figurative constant is used in such a way that the exact number of characters required cannot be determined, only one character is generated. For example, the statement DISPLAY ZEROES would produce one zero character since, in this case, the length of the sequence of zeros to be displayed cannot be determined.

TYPES OF DATA ITEMS

Several types of data items can be described in a COBOL source program. These data items are described in the following text. The format of the Record Description entry used to describe each of these items appears under the discussion of Record Description entries.

Group Items

A group item is defined as one having further subdivisions, so that it contains one or more elementary items. In addition, a group item may contain other groups. An item is a group item if, and only if, its level number is less than the level number of the immediately succeeding item, unless the latter level is 88. If an item is not a group item, then it is an elementary item, or, in the case of level 88, it is a condition-name.

For COBOL F, the maximum length for a group item is 32,767 bytes except for a fixed-length Working-Storage group item, which may be as large as 131,071 bytes.

For COBOL E, the maximum length for any group item or elementary item is 4095 bytes, except for a fixed-length Working-Storage group item, which may be as long as 32,767 bytes.

Elementary Items

Alphabetic Item

An alphabetic item may contain any combination of the characters A through Z and the space. Each alphabetic character is stored in a separate byte.

Alphanumeric Item

An alphanumeric item consists of any combination of characters in the IBM EBCDIC set. Each alphanumeric character is stored in a separate byte.

Report Item

A report item is an alphanumeric item containing only digits and/or special editing characters. It must not exceed 127 characters in length. A report item can be used only as a receiving field for numeric data. Each report character is stored in a separate byte. (See PICTURE and BLANK WHEN ZERO clauses.)

Fixed-Point Items

Fixed-point items may be defined as external decimal, internal decimal, or binary. External decimal corresponds to the form in which information is represented initially for card input, or finally for printed or punched output. Such items may be converted (by moving) to the internal machine formats described as internal decimal or binary. Either of these formats requires less storage than the external decimal format and can be used to save significant amounts of space on input/output units. The binary mode of representation is particularly efficient for data-names used as subscripts. Computational results are the same, regardless of the particular format selected provided the intermediate computational results do not require more than 18 digit positions.

External Decimal Item

Decimal numbers in the System/360 zoned format are external decimal items.Each digit of a number is represented by a single byte, with the four low-order bits of each eight-bit byte containing the value of a digit. The four high-order bits of each byte are zone bits; the zone bits of the low-order byte represent the sign of the item. The maximum length of an external decimal item is 18 digits. For items whose PICTURE does not contain an S, the sign position is occupied by a bit configuration interpreted as positive but which does not represent an overpunch.

Internal Decimal Item

An internal decimal item consists of numeric characters 0 through 9 plus a sign, and represents a value not exceeding 18 digits in length. It appears in storage as "packed" decimal. One byte contains two digits with the low-order byte containing the low-order digit followed by the sign of the item. For items whose PICTURE does not contain an S, the sign position is occupied by a bit configuration interpreted as positive but which does not represent an overpunch.

Binary Item

A binary item may be considered decimally as consisting of numeric characters 0 through 9 plus a sign. It occupies two bytes (a half-word), four bytes (a full-word), or eight bytes (a double word), corresponding to specified decimal lengths of 1 to 4 digits, 5 to 9 digits, and 10 to 18 digits, respectively. The leftmost bit of the reserved area is the operational sign.

Ext Floating Point Items

External and internal floating-point formats define data items whose potential range of value is too great for fixed-point representation. The magnitude of the number represented by a floating-point item must not exceed .72*(10**76).

Ext External Floating-Point Item

An external floating-point item consists of a combination of the characters +, -, blank, decimal point, and digits 0 through 9, appearing in a specific format which represents a number in the form of a decimal number followed by an exponent. The exponent specifies a power of ten

that is used as a multiplier. External floating-point items (also called scientific decimal items) are scanned at object time for conversion to the equivalent internal floating-point value, when used as numeric operands. (See fp-form of PICTURE clause.) Each character of the PICTURE, except E and/or V, represents a single byte of storage reserved for the item.

Internal Floating-Point Item Ext

An internal floating-point item may be considered equivalent to an external floating-point item in capability and purpose. Internal floating-point numbers occupy four or eight bytes, depending on the length of the fractions.

In the short-precision format, the fraction appears in the rightmost three bytes; in the long-precision format, the fraction appears in the rightmost seven bytes. The sign of the fraction is the leftmost bit in either format, and the exponent appears in bit positions 1 through 7.

FILE SECTION

The File Section is used to define data that is contained in files.

FILE AND RECORD HANDLING

For purposes of processing, the contents of a file are divided into logical records. It is important to note that several logical records may occupy a block (i.e., a physical record).

In COBOL there are two classes of files;

1. A file for which there is only one 01-level record description subordinate to the FD entry, called a homogeneous file.

2. A file for which there is more than one 01-level record description subordinate to the FD entry, called a heterogeneous file.

There are also two classes of records that may be contained in a file, fixed-length records and variable-length records. A record is a variable-length record if it contains in its description an OCCURS clause with a DEPENDING ON option; it is fixed-length if it does not.

A homogeneous file may contain either:

1. Fixed—length records, or
2. Variable-length records.

A heterogeneous file may contain records with one of three different characteristics:

1. EQUAL - Each of the records described is fixed in length and all the lengths are equal.

2. DIFFERING - Each of the records described is fixed in length, but at least two record descriptions have differing lengths.

3. VARIABLE - One or more of the records described is variable in length.

Record Types

Three record types are available to COBOL users:

1. Fixed-length (Format F): can be used when all logical records in a file are the same length; may be blocked, and may use the first character to control the printer carriage or punch stacker selection.

2. Variable-length (Format V): records having a count control field; may use the first character to control the printer carriage or punch stacker selection; may be blocked; may have differing or varying lengths.

3. Undefined-length (Format U): unblocked of any length.

Usually, records processed by COBOL object programs are format V records. If the WITHOUT COUNT CONTROL portion of the RECORD CONTAINS clause is written, then either Format U or Format F records are assumed by the compiler. Figure 4 summarizes the IOCS record format types and their relationship to files. The blocking factor is specified by the BLOCK CONTAINS clause.

File Characteristics	Record Length Characteristics	Normal	Without Count Control
Homogeneous	Fixed	V	F
	Variable	V	U
Heterogeneous	Equal	V	F
	Differing	V	U
	Variable	V	U

Figure 4. Record Format Types for Files.

FILE SECTION ENTRIES

Option 1

```
FD file-name VALUE OF FILE-ID IS external-name

[BLOCK CONTAINS integer RECORDS]

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

Ext   [WITHOUT COUNT CONTROL] ]

  [LABEL RECORDS ARE {OMITTED   }]
                     {data-name }

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

F Only [{REPORT IS  }                          ].
       [{REPORTS ARE} report-name...           ]
```

Option 2 F Only

SD sort-file-name VALUE OF FILE-ID IS external-name

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

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

Option 3 F Only

SA saved-area-name

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

Notes

1. The FD entry must describe each data file to be processed by the object program.

2. The SD entry is used to describe each sort-file used in the program. A complete discussion of the Sort feature appears in Section 8. F Only

3. The SA entry is used to define a saved area, and is allowed only in conjunction with a USE FOR RANDOM PROCESSING declarative in the Procedure Division. F Only

4. File-name, sort-file-name, and saved-area-name are highest level qualifiers for their respective record description entries.

5. Each saved-area-name defined by an SA entry must appear in an Option 4 APPLY clause in the Environment Division. F Only

BLOCK CONTAINS Clause

The BLOCK CONTAINS clause specifies the number of logical records of maximum length in a physical record. The format for this clause is:

[BLOCK CONTAINS integer RECORDS]

The BLOCK CONTAINS clause must not be written if the ORGANIZATION clause is specified in the Environment Division, or if U type records are used.

If this clause is omitted, it is assumed that records are not blocked.

RECORD CONTAINS Clause

The RECORD CONTAINS clause is used to specify the maximum size of logical records. The format for this clause is:

```
[RECORD CONTAINS [integer-1 TO] integer-2 CHARACTERS
Ext                    [WITHOUT  COUNT  CONTROL] ]
```

This clause is required if variable length records are described for the file. (i.e., if an OCCURS clause with a DEPENDING ON option is used in the record description). Inclusion of WITHOUT COUNT CONTROL signifies format U records, and hence the BLOCK CONTAINS clause may not be specified for the file.

integer-1 and integer-2 are used to specify minimum and maximum record sizes respectively. If the file contains only fixed-length records, integer-1 (if specified) and integer-2 must be equal to the sizes of the smallest and largest records described for the file respectively. If the file contains variable-length records, integer-1 is ignored and integer-2 is assumed to be the maximum size that any record in the file will have.

The RECORD CONTAINS clause is not necessary for a file having equal length records.

LABEL RECORDS Clause

This clause specifies the presence or absence of user header and trailer labels on a file. This clause is optional. Its format is:

```
[LABEL RECORDS ARE  {OMITTED   }]
                    {data-name }
```

Only the OMITTED option of this clause may be written for files assigned to UNIT-RECORD devices.

For other devices, OMITTED implies either:

a. that there are no user labels or that they are to be bypassed on input.
b. that user labels are not created on output.

Ext Use of the data-name option of this clause means that user labels can be processed (by means of an Option 1 or Option 2 USE section) when a file is opened, closed, or at volume-switching time.

Ext If processing of user labels is desired, the programmer must specify a data-name in the Linkage Section of the Data Division, describing the label. This data-name is then available for reference by a declarative procedure written by the programmer for label processing. Label processing declarative procedures are discussed in Section 6. This data-name may not be subscripted.

'Volume' is a term for a discrete unit on which a file is recorded (e.g., a reel of magnetic tape or a disk pack). In this context, 'volume' and the reserved word UNIT are identical in meaning.

A user label is separate and distinct from a volume label. Volume labels are managed exclusively at the control program level, whereas

user labels are controlled entirely by the user. If compatibility with previous COBOL compilers is desired, this clause is required.

A user label is 80 characters in length. A user header label is characterized by the appearance of UHL in character positions 1 through 3; a user trailer label has UTL in character positions 1 through 3. The remaining 77 characters are formatted according to user choice.

DATA RECORDS Clause

This clause specifies the names of the logical records in a file.

Its format is:

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

This clause is optional; if compatibility with previous COBOL compilers is desired, this clause is required. Record-name is a data-name described with a 01 level-number.

VALUE OF Clause

The VALUE OF clause specifies the name by which the file is known to the Control Program. The format of this clause is:

VALUE OF FILE-ID IS external-name

External-name consists of quotation marks enclosing no more than eight alphabetic and numeric characters, the first of which must be an alphabetic character.

REPORT Clause

F Only

The REPORT clause is required in the File Description entry only when the Report Writer feature is utilized, and the file is either an output report file or is used to contain output report records. A report may be incorporated in more than one file. In this case, the report-name must appear in a REPORT clause of the File Description entry for each file that contains the report.

The format of the REPORT clause is:

```
[{REPORT IS  } report-name...]
 {REPORTS ARE}
```

The appearance of two or more report-names in this clause indicates that the file contains more than one report. These reports may be of different sizes and formats, and the order in which they are described in the Data Division is not significant.

Each report-name listed in the FD entry must have an RD (report description) entry in the Report Section of the Data Division.

Each report-name listed in the REPORT clause of the File Description entry must be associated with a Report Description entry in the Report

Section of the Data Division. (Complete details concerning the Report Writer feature are contained in Section 7.)

The REPORT clause may not be used for files having an ORGANIZATION clause in the Environment Division.

Further information on the REPORT clause is contained in Section 7.

RECORD DESCRIPTION ENTRY

A Record Description entry specifies the characteristics of each item in a data record. Every item must be described in a separate entry in the same order in which the item appears in the record. Each Record Description entry consists of a level-number, a data-name, and a series of independent clauses followed by a period.

The general format of a Record Description entry is:

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

        [PICTURE-clause]  [BLANK-clause]

        [OCCURS-clause]  [VALUE-clause]

        [JUSTIFIED RIGHT]  [SYNCHRONIZED-clause]

        [USAGE-clause] .
```

When this format is applied to specific items of data, it is limited by the nature of the data being described. The allowable format for the description of each data type appears below. Clauses which are not shown in a format are specifically forbidden in that format. Clauses that are mandatory in the description of certain data items are written without brackets.

Group Item

Format:

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

    [OCCURS-clause]  [USAGE-clause]
```

Example:

```
01 GROUP-NAME.
     02 FIELD-B PICTURE X.
```

Elementary Items

Alphabetic Item

Format:

```
level-number {data-name} [REDEFINES-clause] [OCCURS-clause]
             {FILLER   }

PICTURE IS alpha-form [USAGE IS DISPLAY]

[VALUE IS alphabetic-literal] [JUSTIFIED RIGHT] .
```

Example:

```
02 EMPLOYEE-NAME PICTURE A(20) .
```

Alphanumeric Item

Format:

```
level-number {data-name} [REDEFINES-clause] [OCCURS-clause]
             {FILLER   }

PICTURE IS an-form [USAGE IS DISPLAY]

[VALUE IS non-numeric-literal] [JUSTIFIED RIGHT] .
```

Examples:

```
02 MISC-1 PICTURE X(53) .

02 MISC-2 PICTURE XXXXXXXX.
```

Report Item

Format:

```
level-number {data-name} [REDEFINES-clause] [OCCURS-clause]
             {FILLER   }

             {numeric-form BLANK WHEN ZERO }
PICTURE IS   {report-form  [BLANK WHEN ZERO]} [USAGE IS DISPLAY] .
```

Example:

```
02 TOTAL PICTURE $999,999.99-.
```

External Decimal Item

Format:

```
level-number {data-name} [REDEFINES-clause] [OCCURS-clause]
             {FILLER   }
     [VALUE IS numeric-literal].
```

Example:

```
     02 HOURS-WORKED PICTURE 99V9, DISPLAY.
     02 HOURS-SCHEDULED PICTURE 99V9.
```

Internal Decimal Item

Format:

```
level-number {data-name} [REDEFINES-clause] [OCCURS-clause]
             {FILLER   }
     PICTURE IS numeric-form USAGE IS COMPUTATIONAL-3
     [VALUE IS numeric-literal].
```

Example:

```
     02 YEAR-TO-DATE PICTURE S99999999V99 COMPUTATIONAL-3.
```

Binary Item

Format:

```
level-number {data-name} [REDEFINES-clause] [OCCURS-clause]
             {FILLER   }
     PICTURE IS numeric-form USAGE IS COMPUTATIONAL
     [VALUE IS numeric-literal]    [SYNCHRONIZED-clause].
```

Example:

```
     03 SUBSCRIPT PICTURE 999 COMPUTATIONAL.
```

External Floating-Point Item

Ext

Format:

```
level-number {data-name} [REDEFINES-clause] [OCCURS-clause]
             {FILLER   }

     PICTURE IS fp-form [USAGE IS DISPLAY].
```

Example:

```
     02 GAMMA PICTURE +.9(8)E+99.
```

Internal Floating-Point Item

Ext

Format:

```
level-number {data-name} [REDEFINES-clause] [OCCURS-clause]
             {FILLER   }

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

     [VALUE IS floating-point-literal]

     [SYNCHRONIZED-clause].
```

Example:

```
     02 DEVIATION COMPUTATIONAL-1.
```

USAGE Clause

The USAGE clause describes the form in which data is represented.

The USAGE clause may be written at any level. At a group level, it applies to each elementary item in the group. The usage of an elementary item must not contradict the usage explicitly stated for a group to which the item belongs. If USAGE is not specified, the usage of an item is assumed to be DISPLAY. The format of the USAGE clause is:

```
[USAGE IS{DISPLAY      }]
         {COMPUTATIONAL}
         {COMPUTATIONAL-1}
         {COMPUTATIONAL-2}
         {COMPUTATIONAL-3}
```

The DISPLAY option specifies that the item is stored in character form, one character per byte.

The COMPUTATIONAL option specifies a binary data item occupying two, four, or eight character positions corresponding to specified decimal lengths of 1-4, 5-9, and 10-18, respectively. The leftmost bit of the reserved area is the operational sign. Computational items are aligned at the next half, full, or double word boundaries, as appropriate.

The COMPUTATIONAL-1 option specifies a data item stored in short-precision floating-point format.

The COMPUTATIONAL-2 option specifies a data item stored in long-precision floating-point format.

The COMPUTATIONAL-3 option specifies that the item is stored in packed decimal format: two digits per character position, with the low-order half character containing the sign.

PICTURE Clause

The PICTURE clause specifies a detailed description of an elementary level data item and may include specification of special report editing.

The general format of the PICTURE clause is:

Ext

```
PICTURE IS { alpha-form   }
           { an-form      }
           { numeric-form }
           { report-form  }
           { fp-form      }
```

The options are described in the following text.

Alpha-form Option

This option represents an alphabetic item. The PICTURE of an alphabetic item can contain only the character A. An A indicates that the character position will always contain one of the 26 letters of the English alphabet or a space.

An-form Option

This option applies to alphanumeric items. The PICTURE of an alphanumeric item can contain only the character X. An X indicates that the character position will always contain a character from the EBCDIC set.

Numeric-form Option

This option refers to a fixed-point numeric item. The PICTURE of a numeric item may contain a valid combination of the following characters:

CHARACTER	MEANING
9	The character 9 indicates that the actual or conceptual digit position contains a numeric character.
V	The character V indicates the position of an assumed decimal point. Since a numeric item cannot contain an actual decimal point, an assumed decimal point is used to provide the compiler with information concerning the scaling alignment of items involved in computations. Storage is never reserved for the character V.
P	The character P represents a numeric digit position for which storage is never reserved and which always is treated as if it contained a zero. P (or a group of Ps) is used to indicate the location of an assumed decimal point. For example, an item composed of the digits 123 would be treated by an arithmetic procedure statement as 123000 if its PICTURE were 999PPPV; or as .000123 if its PICTURE were VPPP999. The character V may be used or omitted as desired. When used, V must be placed in the position of the assumed decimal point, to the left or right of the P or Ps that have been specified.
S	The character S indicates the presence of an operational sign. If used, it must be the leftmost character of the PICTURE. For a binary item a sign is always present in the item; hence, the presence of S in a numeric-form PICTURE is required. For internal and external decimal items used as input, an S must be written if and only if the item contains a sign. For internal and external decimal items developed by the execution of COBOL statements, the compiler will develop a sign if and only if an S is written in the PICTURE.

Report-form Option

This option refers to a report item. The editing characters that may be combined to describe a report item are: 9 V P . Z * CR DB , 0 B $ + -. The characters 9, P, and V have the same meaning as for a numeric item. The meanings of the other allowable editing characters are described in the following text.

CHARACTER	MEANING
	The decimal point character (.) specifies that an actual decimal point is to be inserted in the indicated position and the source item is to be aligned accordingly. Numeric character positions to the right of an actual decimal point in a PICTURE must consist of characters of one type (i.e., * or Z or 9 or $ or + or -).
Z	The character Z is the zero suppression character. Each Z in a PICTURE represents a digit position. Leading zeros to be placed in positions defined by Z are suppressed, leaving the position blank.

CHARACTER	MEANING
	Zero suppression also terminates upon encountering the actual decimal point (.). The PICTURE ZZZ.ZZ indicates suppression of at most three leading zeroes. It is equivalent to a combination of the BLANK clause and the PICTURE ZZZ.99. Z may appear to the right of an actual decimal point only if all digit positions are represented by Zs. (A Z cannot appear anywhere to the right of a 9).
*	The asterisk is the "check protection" replacement character which is similar to Z, except that leading zeros are replaced by asterisks. If all digit positions are defined by asterisks in the PICTURE, the entire data item will consist of asterisks when the numeric value is zero. An * must not appear anywhere to the right of a 9. The BLANK WHEN ZERO clause may not be applied to an item having an * in its PICTURE.
CR DB	CR and DB are called credit and debit symbols and may appear only at the right end of a picture. These symbols occupy two character positions and indicate that the specified symbol is to appear in the indicated positions if the value of a source item is negative. If the value is positive or zero, spaces will appear instead.
, 0 B	The comma, zero, and B specify insertion of comma, zero, and space, respectively. Each insertion character is counted in the size of the data item, but does not represent a digit position. The presence of zero suppression (Z) or check protection (*) indicates that suppression of leading insertion characters also takes place with associated space or asterisk replacement. These characters may also appear in conjunction with a floating string, as described in the following

A floating string is defined as a leading, continuous series of either $ or + or -, or a string composed of one such character interrupted by Bs and/or commas and/or V or actual decimal point. For example:

```
$$,$$$,$$$
++++
--,---,--
$$$B$$$
+(8)V++
$$,$$$.$$
```

A floating string containing n+1 occurrences of $ or + or - defines n digit positions. When moving a numeric value into a report item, the appropriate character floats from left to right, so that the developed report item has exactly one actual $ or + or - immediately to the left of the most significant nonzero digit, in one of the positions indicated by $ or + or - in the PICTURE. Blanks are placed in all character positions to the left of the single developed $ or + or -. If the most significant digit appears in a position to the right of positions defined by a floating string, then the developed item contains $ or + or - in the rightmost position of the floating string, and nonsignificant zeros may follow. The presence of an actual decimal point in a floating string is treated as if all digit positions to the right of the point were indicated by the PICTURE character 9, and a BLANK WHEN ZERO clause was written for the item. In the following examples, b represents a blank in the developed items.

PICTURE	Numeric Value	Developed Item
$$$999	14	bb$014
--,---,999	-456	bbbbbb-456

A floating string need not constitute the entire PICTURE of a report item, as shown in the preceding examples. Restrictions on characters that may follow a floating string are given later in this description.

When B, comma, or zero appear to the right of a floating string, the string character floats through these characters in order to be as close to the leading digit as possible.

The character B in a floating string indicates that an embedded blank is to appear in the indicated position, unless the position immediately precedes the nonzero, leading significant digit. Embedded Bs in a PICTURE need not be single characters. Thus, $$BB$$$ is a valid PICTURE for a report item. The character comma in a floating string operates similarly, except that the character comma appears in the developed item instead of a blank.

The character V in a floating string serves merely to indicate alignment of the assumed decimal point.

$ + -	The character $ or + or - may appear in a PICTURE either singly or in a floating string. As a fixed sign control character, the + or - must appear as either the first or last symbol in the PICTURE, but not both. The plus sign indicates that the sign of the item is indicated by either a plus or minus placed in the character position, depending on the algebraic sign of the numeric value placed in the report field. The minus sign indicates that blank or minus is placed in the character position, depending on whether the algebraic sign of the numeric value placed in the report field is positive or negative, respectively. As a fixed insertion character, the dollar sign may appear only once in a PICTURE.

Other rules for a report item PICTURE are as follows:

1. The appearance of one type of floating string precludes any other floating string.

2. There must be at least one digit position character.

3. If there are no 9s, BLANK WHEN ZERO is implied unless there is at least one *.

4. The appearance of a floating sign string or fixed plus or minus insertion characters precludes the appearance of any other of the sign control insertion characters, namely, + - CR or DB.

5. The characters in a PICTURE to the left of an actual decimal point (or in the entire PICTURE if no decimal point is given), excluding the characters that comprise a floating string, are subject to the following restrictions:

   a. Z may not follow * or 9 or a floating string.

   b. * may not follow 9 or Z or a floating string.

6. The characters to the right of a decimal point up to the end of a PICTURE, excluding the fixed insertion characters + - CR DB (if present), are subject to the following restrictions:

a. Only one type of digit position character may appear. That is, asterisks, Zs, 9s, and floating string digit position characters $ + - are mutually exclusive.

b. If asterisks appear to the right of a decimal point, then the only allowable digit position character to the left is *. The same is true of Zs.

7. A floating string must begin with at least two consecutive appearances of + or - or $.

8. The PICTURE character 9 can never appear to the left of a floating or replacement character.

9. Floating or replacement characters + - Z $ or * cannot be mixed in a PICTURE description. They may appear with fixed characters as follows:

a. * or Z with fixed $

b. $ (fixed or floating) with fixed rightmost + or -

c. * or Z with fixed leftmost + or -

d. * or Z with fixed rightmost + or -

10. A PICTURE must not begin or end with a comma.

11. If any of the numeric character positions to the right of a decimal point is represented by + or - or $, then all the numeric character positions must be represented by + or - or $.

Ext 12. 9 is the only PICTURE character that may follow a $ - + Z or * appearing to the right of an implied decimal point. Thus, if the value of an item with PICTURE ZZVZ9 is .02, 2 preceded by three blanks will be developed for the item.

Ext <u>Fp-Form Option</u>

This option refers to an external floating-point item. The PICTURE of an external floating-point item consists of all of the following:

1. + or - (+ indicates that a plus sign represents positive values and that a minus sign represents negative values; - indicates that a blank represents positive values and that a minus sign represents negative values).

2. One to sixteen 9s representing mantissa, with a leading or embedded decimal point or V

3. The letter E

4. + or - (see note 1 above)

5. Two 9s representing the exponent

<u>GENERAL NOTES:</u> The following considerations pertain to use of the PICTURE clause.

1. A PICTURE clause must only be used at the elementary level.

2. An integer enclosed in parentheses and following A X 9 Z * 0 P - B $ or + indicates the number of consecutive occurrences of the PICTURE character.

3. All characters, except P V E and S are counted in the total size of a data item. CR and DB occupy two character positions.

4. A maximum of 30 character positions is allowed in a PICTURE character string. For example, PICTURE A(79) consists of five PICTURE characters.

5. A PICTURE must consist of at least one of the characters A X 9 * Z or at least a pair of one of the characters + or - or $.

6. The characters . S V CR and DB can appear only once in a picture. CR and DB may not both appear in the same PICTURE.

7. An item can possess only one sign.

The examples in Figure 5 illustrate the use of PICTURE to edit data. In each example a movement of data is implied, as indicated by the column headings.

Source Area		Receiving Area	
PICTURE	Data	PICTURE	Edited Data
S99999	-12345	-ZZ,ZZ9.99	-12,345.00
S99999V	00123	$ZZ,ZZ9.99	$ 123.00
S9(5)	00100	$ZZ,ZZ9.99	$ 100.00
S9(5)V	-00000	-ZZ,ZZ9.99	- 0.00
9(5)	00000	$ZZ,ZZZ.99	$ .00
9(5)	00000	$ZZ,ZZ.ZZ	
999V99	123.45	$ZZ,ZZ9.99	$ 123.45
V99999	.12345	$ZZ,ZZ9.99	$ 0.12
9(5)	12345	$**,**9.99	$12,345.00
9(5)	00123	$**,**9.99	$***123.00
9(5)	00000	$**,***.99	$******.00
9(5)	00000	$**,***,**	*********
99V999	12.345	$**,**9.99	$****12.34
9(5)	12345	$$$,$$9.99	$12,345.00
9(5)	00123	$$$,$$9.99	$123.00
9(5)	00000	$$$,$$9.99	$0.00
9(4)V9	1234.5	$$$,$$9.99	$1,234.50
V9(5)	.12345	$$$,$$9.99	$0.12
S99999V	-12345	-ZZZZ9.99	-12345.00
S9(5)V	12345	-ZZZZ9.99	12345.00
S9(5)	-00123	-ZZZZ9.99	- 123.00
S99999	12345	ZZZZ9.99-	12345.00
S9(5)	-12345	ZZZZ9.99-	12345.00-
S9(5)	00123	------.99	123.00
S9(5)	-00001	------.99	-1.00
S9(5)	12345	+ZZZZZ.99	+12345.00
S9(5)	-12345	+ZZZZZ.99	-12345.00
S9(5)	12345	ZZZZZ.99+	12345.00+
S9(5)	-12345	ZZZZZ.99+	12345.00-
S9(5)	00123	++++++.99	+123.00
S9(5)	00001	++++++.99	+1.00
9(5)	00123	++++++.99	+123.00
9(5)	00123	------.99	123.00
9(5)	12345	BB999.00	345.00
9(5)	12345	00099.00	00045.00
S9(5)	-12345	$$$$$$.99CR	$12345.00CR
S9(5)	12345	$$$$$$.99CR	$12345.00

Figure 5. Editing Applications of the PICTURE Clause.

BLANK Clause

This clause specifies that the item being described is filled with spaces whenever the value of the item is zero. The BLANK clause may only be used for report items specified at an elementary level.

The format of the BLANK clause is:

[BLANK WHEN ZERO]

VALUE Clause

The VALUE clause defines condition-name values and specifies the initial value of Working-Storage items. The format of this clause is:

[VALUE IS literal]

The size of a literal given in a VALUE clause must be less than or equal to the size of the item as given in the PICTURE clause, with the provision that the literal must also include leading or trailing zeros to reflect Ps in the PICTURE. The positioning of the literal within a data area is the same as the positioning that would result from specifying a MOVE of the literal to the data area. The type of literal written in a VALUE clause depends on the type of item which the clause is used to describe, and is specified in the data item formats earlier in this text.

When an initial value is not specified, no assumption should be made regarding the initial contents of an item in Working-Storage.

The VALUE clause can only be specified for elementary items other than report and external floating point.

In the File Section and Linkage Section the VALUE clause can only appear in conjunction with a level 88 item.

The VALUE clause must not be written in a Record Description entry that also has an OCCURS or REDEFINES clause, or in an entry that is subordinate to an entry containing an OCCURS or REDEFINES clause. In the latter case,an 88 level VALUE clause may be subordinate to the OCCURS or REDEFINES clause. The VALUE clause cannot be used to specify the initial value of an item following a variable portion of a Working-Storage record defined by the DEPENDING option of the OCCURS clause.

REDEFINES Clause

This clause specifies that the same area is to contain different data items, or provides an alternative grouping or description of the same data. The format of the REDEFINES clause is:

data-name-1 [REDEFINES data-name-2]

When written, the REDEFINES clause must be the first clause following the data-name that defines the entry.

When an area is redefined, all descriptions of the area remain in effect. Thus, if B and C are two separate items that share the same storage area due to redefinition, the procedure statements MOVE X TO B or MOVE Y TO C could be executed at any point in the program. In the first case, B would assume the value of X and take the form specified by the description of B. In the second case, the same physical area would receive Y according to the description of C. A redefinition does not cause any data to be erased and does not supersede a previous description.

The REDEFINES clause must not be used for logical records associated with the same file (i.e., it must not be used at the 01 level in the

File Section),since implied redefinition exists for heterogeneous files. The level number of data-name-2 must be identical to that of the item containing the REDEFINES clause. 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 entries giving the new description of the area must immediately follow the entries describing the area being redefined.

A REDEFINES clause may be specified for an item within the scope of an area being redefined; that is, REDEFINES clauses may be specified for items subordinate to items containing REDEFINES clauses.

Except for condition-name entries, entries containing or subordinate to a REDEFINES clause must not contain any VALUE clauses.

The description of data-name-1 or of any item subordinate to data-name-1 may not contain an OCCURS clause with a DEPENDING ON option. Data-name-1 may not be subordinate to an item containing an OCCURS clause. Data-name-2 may not contain an OCCURS clause in its description nor may it be subordinate to an item described by an OCCURS clause. No item subordinate to data-name-2 may be described by an OCCURS clause with a DEPENDING ON option.

Between data-name-2 and data-name-1 there may be no entries having a lower level number than the level number of data-name-2 and data-name-1.

The length of data-name-1, multiplied by the number of occurrences of data-name-1, must be less than or equal to length of data-name-2.

Data-name-2 must not be written with qualifiers. For example, A REDEFINES B OF C is not allowed.

Examples of the REDEFINES clause are contained in Figure 3.

OCCURS Clause

The OCCURS clause is used in defining related sets of repeated data, such as tables, lists, vectors, matrixes, etc. It specifies the number of times that a data item with the same format is repeated. Record Description clauses associated with an item whose description includes an OCCURS clause apply to each repetition of the item being described. When the OCCURS clause is used, the data-name that is the defining name of the entry must be subscripted whenever it appears in the Procedure Division. If this data-name is the name of a group item, then all data-names belonging to the group must be subscripted whenever they are used.

The OCCURS clause must not be used in any Record Description entry having a level number 01 or 88. The OCCURS clause has the following formats:

Option 1

[OCCURS integer TIMES]

In Option 1, integer represents the exact number of occurrences.

Option 2

[OCCURS integer TIMES DEPENDING ON data-name]

In Option 2, integer refers to the maximum number of occurrences. The use of Option 2 does not imply that the length of the data item is variable, but that the number of occurrences of the item may vary. The record containing the variable number of occurrences of the item is, however, of variable length, as is any group containing the variable number of occurrences.

In Option 2, the actual number of occurrences is equal to the value at object-time of the elementary item called data-name. This value must be a positive integer. Hence, the PICTURE for data-name must describe an integer. If data-name appears within the record in which the current Record Description entry also appears, then data-name must precede the variable portion of the record which depends on it. Data-name should be qualified, when necessary, but subscripting is not permitted.

Option 2 may only be used in COBOL E in the following way:

a. only one such clause per logical record is allowed.
b. the clause must appear in the description of either a group that contains the last elementary item of the record, or in the description of the last elementary item itself.
c. the item having an OCCURS clause with a DEPENDING ON option may not itself be contained in a group having any OCCURS clause.

Subscripting

Subscripting provides the facility for referring to data items in a table or list that have not been assigned individual data-names. Subscripting is determined by the appearance of an OCCURS clause in a data description. If an item has an OCCURS clause or belongs to a group having an OCCURS clause, it must be subscripted whenever it is used. A data-name can be subscripted only if the data item is repeated.

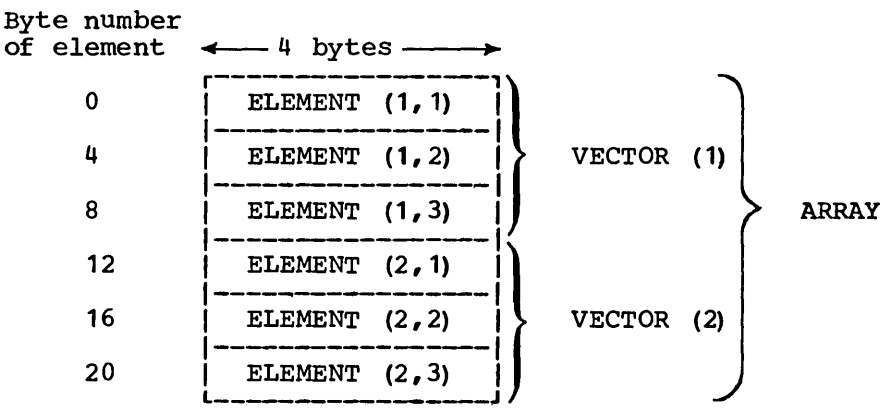
A subscript is a positive nonzero integer whose value determines to which element a reference is being made within a table or list. The subscript may be represented either by a literal or a data-name that has an integral value. Whether the subscript is represented by a literal or a data-name, the subscript is enclosed in parentheses and appears after the terminal space of the name of the element.

Tables may be defined so that more than one level of subscripting is required to locate an element within them. Such a case exists when a group item described with an OCCURS clause contains one or more items also described with OCCURS clauses. A maximum of three levels of

subscripting is permitted by COBOL. Multilevel subscripts are always written from left to right, in decreasing order of inclusiveness of the groupings in the table. Subscripts are written within a single pair of parentheses and are separated by a comma followed by a space. For example:

```
01     ARRAY.
  02     VECTOR, OCCURS 2 TIMES.
    03     ELEMENT, OCCURS 3, PICTURE S9(9),
           USAGE IS COMPUTATIONAL.
```

The above example would be allocated storage as follows:



A data-name may not be subscripted under the following circumstances:

1. When it is being used as a subscript.
2. When it is being used as a qualifier.
3. When it appears as the defining name of a record description entry.
4. When it appears as data-name-2 in a REDEFINES clause.
5. When it appears as data-name in the DEPENDING ON option of the OCCURS clause.
6. When it is data-name in a SYMBOLIC KEY or in an ACTUAL KEY clause.
7. When it is data-name in a LABEL RECORDS clause.

JUSTIFIED RIGHT Clause

This clause may be written only for an elementary alphabetic or alphanumeric item. Its format is:

[JUSTIFIED RIGHT]

When non-numeric data is moved to a field for which JUSTIFIED RIGHT has been specified, the rightmost character of the source field is placed in the rightmost position of the receiving field. The moving of characters continues from right to left until the receiving field is filled. If the length of the source field is greater than that of the

receiving field, truncation terminates the move after the leftmost position of the receiving field is filled. If the source field is shorter, the remaining leftmost positions of the receiving field are filled with spaces.

SYNCHRONIZED Clause

The SYNCHRONIZED clause is never necessary in a System/360 COBOL program since binary and floating-point data items are automatically synchronized. If the clause is specified, it is treated as comments by the compiler.

The compiler assigns storage so that the starting byte of a binary or internal floating-point item is on the next available half-word, full-word, or double-word boundary, as appropriate. In this way, an internal floating-point or binary item is properly aligned at the storage location required by the computer.

The format of the SYNCHRONIZED clause is:

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

If a data hierarchy contains binary or floating point items intermixed with other elementary items, there may exist "slack bytes" introduced to assure the necessary byte alignment (implicit synchronization). The following machine requirments pertain to binary and floating point items:

ITEM	Must begin at a byte address address that is a multiple of
Binary half-word	k = 2
Binary full-word	k = 4
Binary double-word	k = 4
Floating point, short	k = 4
Floating point, long	k = 8

When an item of the types listed in the above chart immediately follows an item whose last byte is at address X, relative to the first byte of the COBOL record, then the first byte of this item is at address Z, where Z is computed as follows:

$Z = k * y$, where y is the truncated integral value of $\frac{X + k.}{k}$

Note that k is taken from the preceding chart.

The number of slack bytes is Z - X - 1, which may be zero or positive and non-zero, depending on X and k.

Slack bytes exist in a record not only in main storage but on files. The compiler inserts slack bytes on output and expects them on input.

WORKING-STORAGE SECTION

The Working-Storage Section is used to describe areas of storage reserved for intermediate processing of data. This section consists of a series of Record Description entries, each of which describes an item in a work area.

An independent Working-Storage entry describes a single item that is not subdivided and is not itself a subdivision of some other item. Each of these items is defined in a separate Record Description entry, which begins with the special level number 77. All independent Working-Storage entries must precede any items having any of the level numbers 01 through 49.

Data items in the Working-Storage Section that bear a definite relationship to each other must be grouped into records according to the rules for formation of record descriptions. All clauses that are used in Record Description entries may be used in Working-Storage record descriptions. Each data-name in the Working-Storage Section that identifies a record (01 or 77 level) must be unique, since it cannot be qualified by a file-name. Subordinate data-names need not be unique, if they can be made unique by qualification.

In Working-Storage, level 01 items are adjusted to a double-word boundary; level 77 binary or internal floating-point items are adjusted to the next available half-word, full-word, or double-word boundary, as appropriate.

No assumption should be made about the initial values of Working-Storage items when these items have not had their initial values defined in a VALUE clause.

Ext

LINKAGE SECTION

The Linkage Section describes data passed from another program, or user label record areas.

Record description entries in the Linkage Section provide names and descriptions but storage within the program is not reserved, since the data exists elsewhere. Any Record Description clause may be used to describe items in the Linkage Section, with one exception: the VALUE clause may not be specified for other than level 88 items.

The Linkage Section is required in any program in which a LABEL RECORDS clause with a data-name option or an ENTRY statement with a USING option appears. A complete discussion of ENTRY is contained in Section 6.

The Procedure Division of a source program specifies those procedures needed to solve a given problem. These steps (computations, logical decisions, input/output, etc.) are expressed in meaningful statements, similar to English, which employ the concept of verbs to denote actions, statements and sentences to describe procedures. The Procedure Division must begin with the header PROCEDURE DIVISION followed by a period.

SYNTAX

The discussion that follows describes the units of expression that constitute the Procedure Division and the way in which they may be combined. The smallest unit of expression in the Procedure Division is the statement. Sentences, paragraphs, and sections are the larger units of expression.

STATEMENTS

A statement consists of a COBOL verb or the word IF or ON, followed by any appropriate operands (data-names, file-names, or literals) and other COBOL words that are necessary for the completion of the statement. The three types of statements are: compiler-directing, imperative, and conditional.

Compiler-Directing Statement

A compiler-directing statement directs the compiler to take certain actions at compilation time. A compiler-directing statement contains one of the compiler-directing verbs (ENTER, EXIT, NOTE) and its operands. Compiler-directing statements (except for NOTE) must appear as separate single sentences.

Imperative Statement

An imperative statement specifies an unconditional action to be taken by the object program. An imperative statement consists of a COBOL verb and its operands, excluding the Compiler-Directing verbs and the conditional statements. An imperative statement may also consist of a series of imperative statements.

Conditional Statement

A conditional statement is a statement containing a condition that is tested to determine which of alternate paths of program flow is to be taken.

The following are conditional statements:

1. A READ statement
F Only 2. A RETURN statement in the Sort Feature
3. A WRITE statement with the INVALID KEY option
Ext 4. A REWRITE statement with the INVALID KEY option
5. An arithmetic statement with the SIZE ERROR option
Ext 6. An ON statement
7. An IF statement

Although IF and ON are not verbs in the grammatical sense, they are regarded as such in COBOL, inasmuch as they are the key words associated with a particular statement form.

The conditions evaluated in conditional statements are:

1. AT END or INVALID KEY in a READ or RETURN statement
2. INVALID KEY in a WRITE or REWRITE statement
3. SIZE ERROR in an arithmetic statement
4. The count-condition in an ON statement
5. One of four tests in an IF statement

The conditions in 1 to 4 above are called 'event-conditions.' The conditions in 5 above are called 'test-conditions.'

The formats for the conditions named in 1 to 4 above are discussed in the text for their respective statements. The types of conditions evaluated in an IF statement are discussed in the section "Test-Conditions."

SENTENCES

A sentence is a single statement or a series of statements terminated by a period and followed by a space. A single comma or semicolon may be used as a separator between statements. A sentence must not begin in margin A.

PARAGRAPHS

Paragraphs are logical entities consisting of one or more sentences. Each paragraph may begin with a paragraph-name or section-name.

Paragraph-names and section-names are procedure-names. Procedure-names follow the rules for word formation. Therefore, they may be composed solely of numeric characters and are equivalent only if they are composed of the same number of numeric digits and have the same numeric value. Thus, 0023 is not equivalent to 23.

A paragraph need not begin with a paragraph-name in the Declaratives section of the Procedure Division or with a paragraph-name or section-name after the words PROCEDURE DIVISION or END DECLARATIVES. In the Declaratives section a paragraph begins after the USE sentence or after a paragraph-name.

A paragraph-name must not be duplicated within the same section. When used as operands in Procedure Division statements, nonunique paragraph-names may be uniquely qualified by writing IN or OF after the paragraph-name, followed by the name of the section in which the

paragraph is contained. A paragraph-name need not be qualified when referred to from within the section in which it is contained. A paragraph ends at the next paragraph-name or section-name, or at the end of the Procedure Division. In the case of Declaratives, a paragraph ends at the next paragraph-name, section-name, or at END DECLARATIVES.

SECTIONS

A section is composed of one or more successive paragraphs and must begin with a section-header. A section-header consists of a section-name conforming to the rules for procedure-name formation, followed by the word SECTION and a period. A section header must appear on a line by itself, except in the Declaratives portion of the Procedure Division, where it may only be followed immediately by a USE sentence or an INCLUDE statement. The INCLUDE statement is discussed in Section 9.

A section ends at the next section-name or at the end of the Procedure Division, or, in the case of Declaratives, at the next section-name or at END DECLARATIVES.

IF STATEMENT

For COBOL F the format of the IF statement is:

```
IF condition [THEN] {statement-1...   }
                    {NEXT SENTENCE    }

    [{ELSE     } {statement-2...   }]
     {OTHERWISE} {NEXT SENTENCE    }
```

For COBOL E, only imperative statements can follow the condition. Therefore the format for the IF statement for COBOL E is:

```
IF condition [THEN] {imperative-statement...}
                    {NEXT SENTENCE          }

    [{ELSE     } {statement-2...   }]
     {OTHERWISE} {NEXT SENTENCE    }
```

ELSE (or OTHERWISE) NEXT SENTENCE may be omitted if it immediately precedes the period for the sentence.

EVALUATION OF CONDITIONAL STATEMENTS

When a condition is evaluated the following action is taken:

1. If the condition is true, the statements immediately following the condition are executed.

2. If the condition is false and the conditional statement is an IF or ON statement, the statements following ELSE or OTHERWISE (or the next sentence) are executed.

The AT END, INVALID KEY, and SIZE ERROR conditions are followed by a series of imperative statements. In an ON count-conditional statement, the count-condition is followed by a series of imperative statements (or

NEXT SENTENCE) and may be followed by the words ELSE or OTHERWISE followed by a series of statements (or NEXT SENTENCE). The formats of the IF statement describe what may follow the condition in the IF statement.

A series of imperative statements is terminated by one of the following:

1. A period.

2. ELSE or OTHERWISE associated with a previous IF or ON.

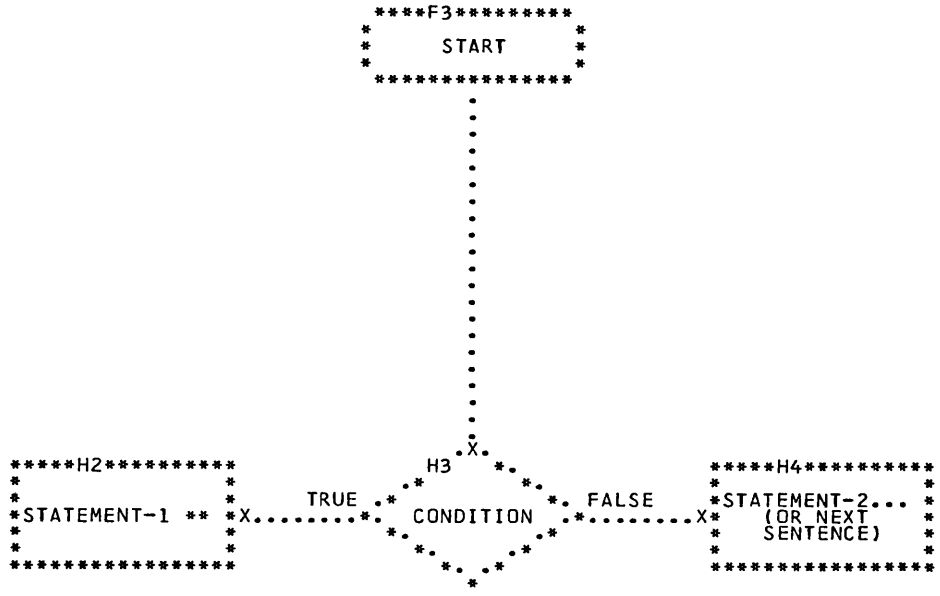
In a series of imperative statements executed if a condition is true, only the last statement may be an Option 1 GO TO statement or a STOP RUN statement; otherwise the series of statements would contain statements to which control cannot flow.

For example, in the following paragraph, the statement MOVE A TO B could never be executed whether or not the AT END condition were found to be false.

```
W.  READ PAYROLL-RECORD AT END GO TO Y MOVE A TO B.
```

Figure 6 is a flowchart showing how an IF or ON conditional statement is evaluated.

Figure 7 is a flowchart showing how a conditional statement other than IF or ON is evaluated.



** For COBOL E, statement-1 is one or more imperative-statements. For COBOL F, statement-1 is one or more statements.

Figure 6. Evaluation of IF or ON Conditional Statement.

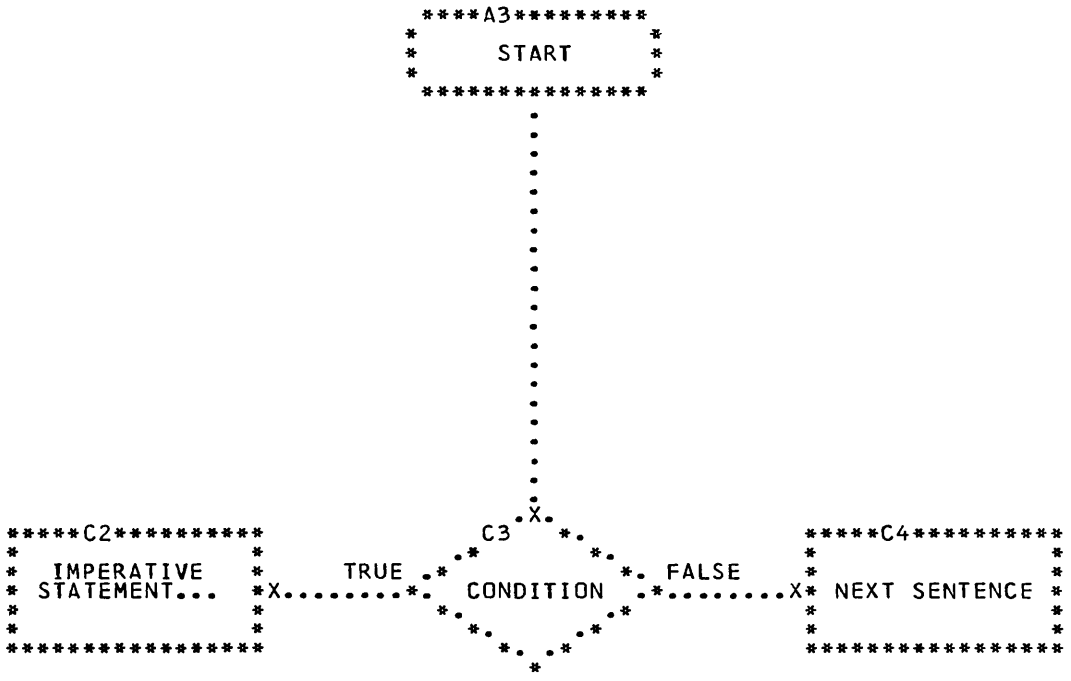


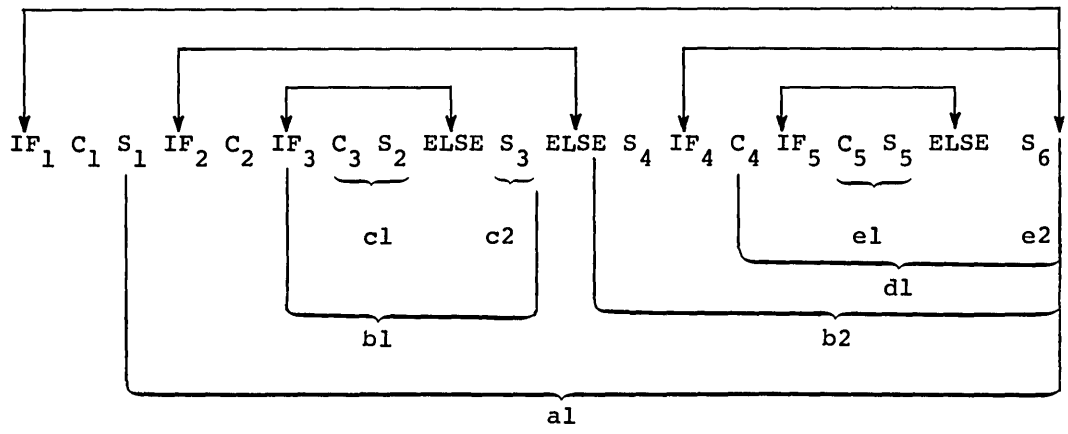
Figure 7. Evaluation of Conditional Statement other than IF or ON.

NESTED IF STATEMENTS F Only

Statement-1 and statement-2 in IF statements 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. Nesting statements is much like specifying subordinate arithmetic expressions enclosed in parentheses and combined in larger arithmetic expressions.

IF statements contained within IF statements must be considered as paired IF and ELSE combinations, proceeding from left to right. Thus, any ELSE encountered must be considered to apply to the immediately preceding IF that has not already been paired with an ELSE. In the conditional statement in Figure 8, C stands for condition; S stands for any number of imperative statements; and the pairing of IF and ELSE is shown by the lines connecting them.

Figure 9 is a flowchart indicating the logical flow of the conditional statement in Figure 8.



a1 - Statement-1 for IF_1

(If C_1 is false, the next sentence is executed, since there is no ELSE for it.)

b1 - Statement-1 for IF_2

b2 - Statement-2 for IF_2

c1 - Statement-1 for IF_3

c2 - Statement-2 for IF_3

d1 - Statement-1 for IF_1

(If C_4 is false, the next sentence is executed, since there is no ELSE for it.)

e1 - Statement-1 for IF_5

e2 - Statement-2 for IF_5

Figure 8. Conditional Statements with Nested IF Statements.

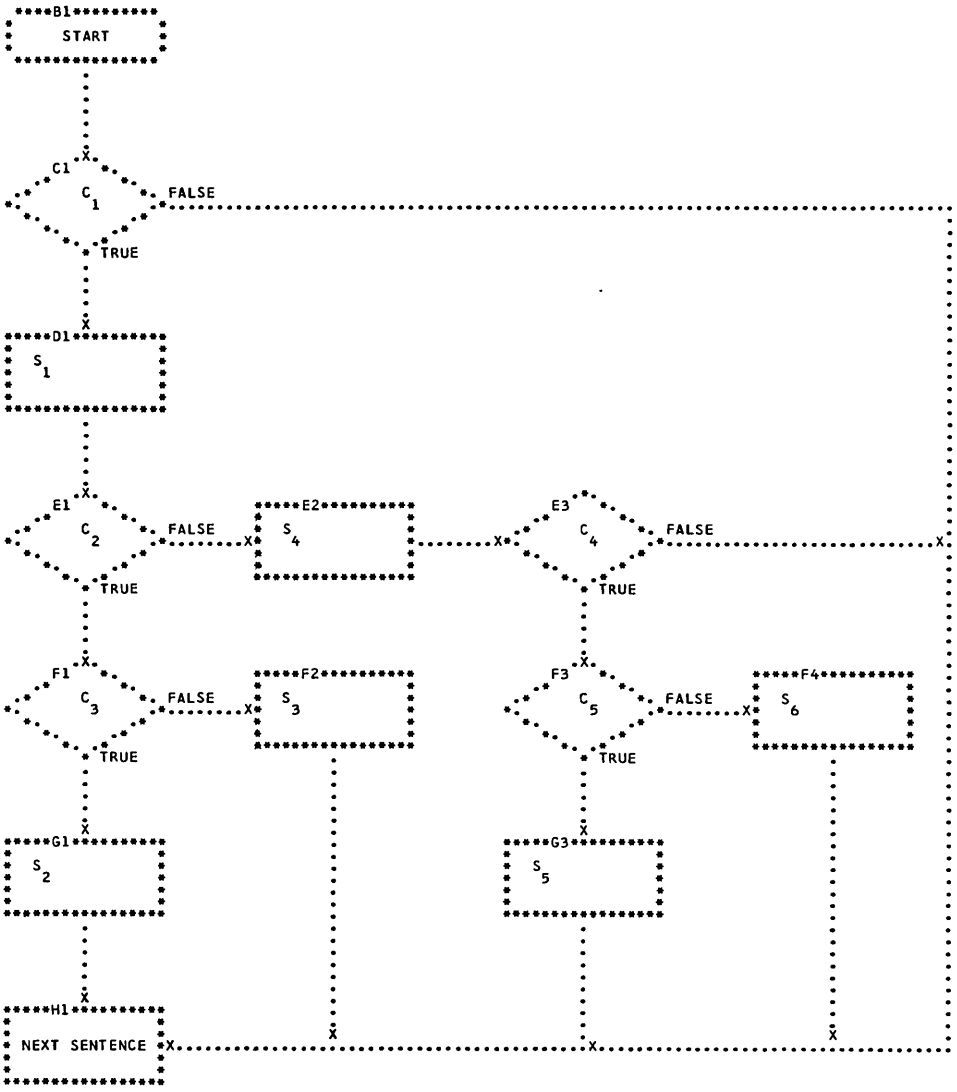


Figure 9. Logical Flow of Conditional Statement with Nested IF Statements.

TEST-CONDITIONS

A test-condition is an expression that, taken as a whole, may be either true or false, depending on the circumstances existing when the expression evaluated.

There are four types of simple test-conditions. When preceded by the word IF, each constitutes one of four types of tests: relation test, sign test, class test, condition name test.

The word NOT may be used in any simple test-condition to make the relation specify the opposite of what it would express without the word NOT. For example, AGE NOT GREATER THAN 21 is the opposite of AGE GREATER THAN 21. NOT may also precede an entire condition, as in NOT (AGE GREATER THAN 21). AGE NOT GREATER THAN 21 and NOT (AGE GREATER THAN 21) are identical in meaning.

Relation Test

A relation test involves the comparison of two operands, either of which can be a data-name, a literal, or an arithmetic expression. The comparison of two literals is not permitted. A figurative constant may be used instead of either literal-1 or literal-2 in a relation test.

The format for a relation test is:

```
{data-name-1              }
{arithmetic-expression-1  }
{figurative-constant-1    } IS [NOT] {>                 }
{literal-1                }          {<                 }
                                     {=                 }
                                     {GREATER THAN      }
                                     {LESS THAN         }
                                     {EQUAL TO          }

    {data-name-2              }
    {arithmetic-expression-2  }
    {figurative-constant-2    }
    {literal-2                }
```

The symbol > is equivalent to the reserved words GREATER THAN. The symbol < is equivalent to the reserved words LESS THAN. The equal sign is equivalent to the reserved words EQUAL TO.

COMPARISON OF NUMERIC ITEMS: For numeric items, a relation test determines that the value of one of the items is less than, equal to, or greater than the other, regardless of the length. Numeric items are compared algebraically after alignment of decimal points. Zero is considered a unique value, regardless of length, sign, or implied decimal-point location of an item.

COMPARISON OF NON-NUMERIC ITEMS: For non-numeric items, a comparison results in the determination that one of the items is less than, equal to, or greater than the other, with respect to the binary collating sequence of characters in the IBM Extended BCD Interchange Code.

If the non-numeric items are of the same length, the comparison proceeds by comparing characters in corresponding character positions, starting from the high-order position and continuing until either a pair of unequal characters or the low-order position of the item is compared. The first pair of unequal characters encountered is compared for relative position in the collating sequence. The item containing the character that is positioned higher in the collating sequence is the greater item. The items are considered equal after the low-order position is compared.

If the non-numeric items are of unequal length, comparison proceeds as described for items of the same length. If this process exhausts the characters of the shorter item, the shorter item is less than the longer, unless the remainder of the longer item consists solely of spaces, in which case, the items are equal. If desired, the characters of an item may be altered prior to a comparison by use of a TRANSFORM statement in order to reflect another collating sequence.

Figure 10 indicates the characteristics of the items being compared and the type of comparison made. A blank box in Figure 10 indicates that the test is not permitted.

		SECOND OPERAND									
		GR	AL	AN	ED	ID	BI	EF	IF	RP	FC
FIRST OPERAND	Group Item (GR)	NN	NN	NN	NN	NN	NN	NN	NN	NN	NN
	Alphabetic Item (AL)	NN	NN	NN						NN	NN*
	Alphanumeric (non-report) Item (AN)	NN	NN	NN						NN	NN
	External Decimal Item (ED)	NN			NU	ID	ID	IF	IF		ZE
	Internal Decimal Item (ID)	NN			ID	NU	ID	IF	IF		ZE
	Binary Item (BI)	NN			ID	ID	NU	IF	IF		ZE
	External Floating-point Item (EP)	NN			IF	IF	IF	NU	IF		ZE
	Internal Floating-point Item (IP)	NN			IF	IF	IF	IF	NU		ZE
	Report Item (RP)	NN	NN	NN						NN	NN
	Figurative Constant (FC)	NN	NN*	NN	ZE	ZE	ZE	ZE	ZE	NN	

*Permitted with the figurative constants SPACE and ALL 'character', where character must be alphabetic.

Abbreviations for Types of Comparison

NN Comparison as described for non-numeric items.
NU Comparison as described for numeric items.
IF Comparison as described for numeric items.
If an item is not internal floating point, it is converted to internal floating point before comparison.
ID Comparison as described for numeric items.
If an item is not internal decimal, it is converted to internal decimal before comparison.
ZE Valid only is figurative constant is ZERO.

Figure 10. Permissible Comparisons.

Sign Test

This type of condition tests whether or not the value of a numeric item is less than zero (NEGATIVE), greater than zero (POSITIVE), or is zero (ZERO). The value zero is considered neither positive nor negative.

The format for a sign test is:

```
{data-name              }
{arithmetic-expression  } IS [NOT] {POSITIVE}
                                   {ZERO    }
                                   {NEGATIVE}
```

Class Test

When a class test is specified, determination is made as to whether or not an item consists solely of the following:

1. The characters 0 through 9 (NUMERIC)
2. The characters A through Z and space (ALPHABETIC)

The item to be tested can be elementary alphanumeric, alphabetic, internal decimal, or external decimal. The valid forms of the class test are shown in Figure 11.

The format for the class test is:

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

If the last character of an otherwise numeric field contains a digit with a sign over punch, the field is considered numeric. For a single character alphanumeric field containing a digit with a sign overpunch, the tests IF NUMERIC and IF ALPHABETIC will both be considered true while the NOT form of the tests will both be false.

Type of Item	Only Valid Forms of Class Test	
Alphabetic	ALPHABETIC	NOT ALPHABETIC
Alphanumeric	ALPHABETIC	NOT ALPHABETIC
	NUMERIC	NOT NUMERIC
Internal or External Decimal	NUMERIC	NOT NUMERIC

Figure 11. Valid Forms of Class Test

Condition-name Test

The format for condition-name test is:

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

A condition-name test is one in which a conditional variable is tested to see whether or not its value is greater than, equal to, or less than the value specified for a condition-name associated with it. For example, in a program processing a payroll, the data item MARITAL-STATUS (the conditional variable) might be a code indicating whether an employee is married, divorced, or single. Assume that if MARITAL-STATUS has the value of 1, the employee is single; if it has the value of 2, he is married; and if it has the value of 3, he is divorced. To determine whether or not an employee is married, the programmer could test this condition by using a simple relational condition in a conditional statement such as IF MARITAL-STATUS = 2 SUBTRACT MARRIED-DEDUCTION FROM GROSS. Alternatively, he can associate a condition-name with each value that MARITAL-STATUS might assume. Thus, in the Data Division, the condition-names SINGLE, MARRIED, and DIVORCED might be associated with values 1, 2, and 3, respectively. For example:

```
02 MARITAL-STATUS PICTURE 9.
88 SINGLE VALUE IS 1.
88 MARRIED VALUE IS 2.
88 DIVORCED VALUE IS 3.
```

Then, as a shorthand form of the simple relational condition MARITAL-STATUS = 1, the programmer could write the single condition-name SINGLE. Therefore, the following two statements would produce identical results:

```
IF MARITAL-STATUS = 1 GO TO Z.
IF SINGLE GO TO Z.
```

The condition-name test, then, is an alternative way of expressing certain conditions which could be expressed by a simple relational condition.

COMPOUND CONDITIONS

Simple test-conditions can be combined with logical operators according to specified rules to form compound conditions. The logical operators are AND, OR, and NOT. Two or more simple conditions combined by AND and/or OR make up a compound condition.

The word OR is used to mean either or both. Thus, the expression A OR B is true if: A is true, B is true, or both A and B are true. The word AND is used to mean both. Thus, the expression A AND B is true only if both A and B are true. The word NOT is used in the manner described in the subsection "Test-Conditions." Thus, the expression NOT (A OR B) is true if A and B are false; and the expression NOT (A AND B) is true if A is false, B is false, or if both A and B are false.

The logical operators and truth values are shown in Figure 12, where A and B represent simple test-conditions.

Condition		Related Conditions				
A	B	NOT A	A AND B	A OR B	NOT (A AND B)	NOT (A OR B)
True	True	False	True	True	False	False
False	True	True	False	True	True	False
True	False	False	False	True	True	False
False	False	True	False	False	True	True

Figure 12. Truth Table.

Parentheses may be used to specify the order in which conditions are evaluated. Parentheses must always be paired. Logical evaluation begins with the innermost pair of parentheses and proceeds to the outermost. If the order of evaluation is not specified by parentheses, the expression is evaluated in the following way:

1. AND and its surrounding conditions are evaluated first, starting at the left of the expression and proceeding to the right.
2. OR and its surrounding conditions are then evaluated, also working from left to right.

Thus, the expression: A IS GREATER THAN B OR A IS EQUAL TO C AND D IS POSITIVE would be evaluated as if it were parenthesized as follows:

```
(A IS GREATER THAN B) OR ((A IS EQUAL TO C)
AND (D IS POSITIVE)).
```

The rules for formation of symbol pairs are shown in Figure 13. The letter C stands for conditional expression. P means that the combination is permissible. A dash means that the combination is not permissible.

First Symbol	Second Symbol						
		C	OR	AND	NOT	(	)
	C	-	P	P	-	-	P
	OR	P	-	-	P	P	-
	AND	P	-	-	P	P	-
	NOT	P	-	-	-	P	-
	(	P	-	-	P	P	-
	)	-	P	P	-	-	P

Figure 13. Formation of Symbol Pairs

F Only IMPLIED SUBJECTS AND OPERATORS

Simple relation test test-conditions may have implied first operands (subjects) when combined to form compound conditions. The following is the format for a series of relation tests forming a compound condition with implied first operands. The relational operators are GREATER THAN, LESS THAN, >, etc.

operand-1 IS [NOT] relational-operator operand-2

$$\left\{\begin{matrix}\underline{\text{AND}}\\ \underline{\text{OR}}\end{matrix}\right\} \quad [\underline{\text{NOT}}] \text{ relational-operator operand-3...}$$

Thus, the following statement could be made:

```
IF ACCOUNT-NUMBER IS GREATER THAN COUNT-A AND NOT LESS THAN COUNT-B OR =
COUNT-C GO TO Z.
```

A relational operator can be implied only when a first operand is also implied. The following is the format for a series of relation tests forming a compound condition with implied first operand and relational operators.

operand-1 IS [NOT] relational-operator

$$\text{operand-2}\left\{\begin{matrix}\underline{\text{AND}}\\ \underline{\text{OR}}\end{matrix}\right\} \quad [\underline{\text{NOT}}] \text{ operand-3...}$$

Thus, the following statement could be made:

```
IF ACCOUNT-NUMBER GREATER THAN COUNT-A AND COUNT-B OR COUNT-C GO TO Z.
```

ARITHMETIC EXPRESSIONS

An arithmetic expression consists of arithmetic operators, data-names, and/or literals representing items on which arithmetic may be performed.

The following five arithmetic operators may be used in arithmetic expressions:

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

Parentheses may be used to indicate the hierarchy of operations on elements in an arithmetic expression.

When the hierarchy of operations in an expression is not completely specified by parentheses, the order of operations is assumed to be exponentiation, then multiplication and division, and finally addition and subtraction. Thus, the expression A+B/C+D**E*F-G is taken to mean A+(B/C)+((D**E)*F)-G.

When the order of a sequence of consecutive operations on the same hierarchical level (i.e., consecutive multiplications and divisions or consecutive additions and subtractions) is not completely specified by parentheses, the order of operation is assumed to be from left to right. Thus, certain expressions ordinarily considered ambiguous are permitted in COBOL. For example, A/B*C and A/B/C are taken to mean (A/B)*C and (A/B)/C, respectively. The expression A*B/C*D is taken to mean ((A*B)/C)*D.

One exception should be noted, however. An expression having a succession of exponentiation operators separating a series of operands is evaluated as though parenthesization were inserted from the right. Thus, A**B**C is evaluated as though it were written (A**(B**C)).

Exponentiation of a negative value is allowed only if the exponent is a literal or data-name having an integral value.

Exponentiation is performed in floating-point when an exponent is a fractional literal or is a data-name whose PICTURE describes a fractional number.

The minus sign is the only allowable unary operator (having only one operand). The unary minus sign must be the first character of an arithmetic expression or must be immediately preceded by a left parenthesis.

COMPILER-DIRECTING DECLARATIVES

Declarative sections are identified by compiler-directing statements that specify the circumstances under which a procedure is to be executed in the object program. Declaratives consist of a section-name, followed by the word SECTION and a period, and a USE sentence followed by procedural statements. Declarative sections must be grouped together at the beginning of the Procedure Division, preceded by the key word

DECLARATIVES in Margin A, and followed by the key words END DECLARATIVES, where END must also appear in Margin A. DECLARATIVES and END DECLARATIVES must be followed by a period.

The general form for declaratives is:

```
PROCEDURE DIVISION.
DECLARATIVES.
 {section-name SECTION. USE-sentence.
 {[paragraph-name.] sentence... .] ...] ...
END DECLARATIVES.
```

The occurrence of another section or the words END DECLARATIVES terminates a previous USE section. If there are two or more logical paths within a declarative procedure, these paths must lead to a common path within the section containing them. An ALTER, PERFORM, or GO TO statement within a declarative section must not refer to paragraph-names or section-names outside that declarative section, except that a GO TO statement in an Option 1 or Option 2 USE section may refer to the reserved word MORE-LABELS.

A declarative section may not be referred to by any PERFORM or GO TO statement outside the declarative. Within a given declarative section, there may be no reference to a point outside the declarative.

USE

The USE sentence identifies the type of declarative.

There are four options of the USE sentence. Each identifies the following types of procedures.

1. Label-checking procedures
2. Label-writing procedures
3. Asynchronous processing procedures
4. Report-writing procedures

The formats of the USE sentence are:

Ext Option 1

```
USE FOR CREATING [BEGINNING]
                 [ENDING   ]

    LABELS ON OUTPUT [file-name...].
```

Ext Option 2

```
USE FOR CHECKING [BEGINNING]
                 [ENDING   ]

    LABELS ON INPUT [file-name...].
```

Options 1 and 2 are used to provide user label processing procedures. CHECKING refers to an input file; CREATING refers to an output file. In this context, 'input' means all files opened as INPUT or I-O.

The word BEGINNING refers to user header labels; the word ENDING refers to user trailer labels. Absence of either word indicates that the USE section will process both headers and trailers.

The exit from an Option 1 or Option 2 USE section is inserted by the compiler following the last statement in the section. All logical program paths within the section must lead to this point. One exception to this rule is allowed: a special exit may be specified by the statement GO TO MORE-LABELS. When an exit is made from a label processing USE section by means of this statement, IOCS is directed to do one of the following:

a. read an additional user header label or user trailer label and then re-enter the USE section for further checking of labels. In this case, IOCS will only re-enter the USE section if there exists another user label to check. Hence, there need not exist a program path that flows through the last statement in the section. The point of return to the USE section, after exit by means of a GO TO MORE-LABELS statement, is the beginning of the section.

b. write the current user header label or user trailer label and then re-enter the USE section for further creating of labels.

If no GO TO MORE-LABELS statement is executed, then the USE section is not reentered to check or create any immediately succeeding user labels.

The user label is contained in an IOCS area. If label processing is desired, the label must be described as a data item in the Linkage Section of the Data Division and must be listed as a data-name in the LABEL RECORDS clause in the File Description entry for the file.

In case 'b' above, a label is written each time that an exit from the USE section takes place. The label is created in an IOCS area.

In an Option 1 USE section, there must be a path of program flow through the last statement of the section, so that writing of user labels can be terminated.

Option 3 F Only

USE FOR RANDOM PROCESSING.

Option 3 is used to specify out-of-line procedural statements for asynchronous processing. STOP, OPEN, and CLOSE statements are not allowed in out-of-line procedures.

Option 4 F Only

USE BEFORE REPORTING data-name-1.

Option 4 is used to designate procedures to be executed by the Report Writer before the Report group specified by data-name-1 is produced; data-name-1 may be the name of any type of report group except DETAIL.

Report Writer verbs may not be used in procedures associated with the USE option. Further information on the Report Writer is contained in Section 7.

The COBOL verbs are the basis of the Procedure Division of a source program.

The organization of the remainder of this section is based on the classifications used in the following list:

Input/Output Verbs
- OPEN
- READ
- WRITE
- Ext REWRITE
- CLOSE
- ACCEPT
- DISPLAY

F Only Asynchronous Processing Verbs
- PROCESS
- HOLD

Data Manipulation Verbs
- MOVE
- EXAMINE
- Ext TRANSFORM

Arithmetic Verbs
- COMPUTE
- ADD
- SUBTRACT
- MULTIPLY
- DIVIDE

Procedure-Branching Verbs
- STOP
- GO TO
- ALTER
- PERFORM

Compiler-Directing Verbs
- EXIT
- ENTER
- NOTE

INPUT/OUTPUT STATEMENTS

OPEN

The OPEN statement initiates the processing of files. When applicable, execution of an OPEN statement initiates label checking for input and output files, and label creation for output files. At this time, appropriate label-handling procedures specified by a USE declarative are executed.

The format of an OPEN statement is:

```
OPEN  {INPUT     file-name  [WITH NO REWIND]  [REVERSED]}
      {OUTPUT    file-name  [WITH NO REWIND]            }
      {I-O       file-name                              }
      [INPUT     file-name  [WITH NO REWIND]  [REVERSED]]
      [OUTPUT    file-name  [WITH NO REWIND]            ] ...
      [I-O       file-name                              ]
```

The OPEN statement must be executed prior to any other input/output statement for any file. The OPEN statement, by itself, does not make an input record available for processing; a READ statement must be executed to obtain the first data-record. For an output file, an OPEN statement makes available an area for development of the first output record. A second OPEN statement for a given file cannot be executed prior to the execution of a CLOSE statement for that file.

The I-O option permits the opening of a direct-access file for both input and output operations.

An OPEN statement for an I-O file performs the same label checking functions as for an input file.

The NO REWIND option should only be written for files assigned to UTILITY device-numbers for which rewinding is possible, e.g., 2400, 7340, etc. This option suppresses the rewinding normally associated with opening a file.

The REVERSED option can only be applied to files assigned to specific devices for which the reverse-read feature is available. If the device-number is not written in the ASSIGN clause of the Environment Division, the REVERSED option is not permitted. The REVERSED option may not be used for a file containing blocked type V records.

An example of the OPEN statement is:

```
OPEN INPUT X-FILE, OUTPUT Y-FILE Z-FILE NO REWIND INPUT R-FILE NO
REWIND REVERSED.
```

Note that Y-FILE is not opened with no rewind.

READ

The functions of the READ statement are:

1. For sequential file processing, to make available the next logical record from an input file and to allow performance of specified imperative statements when end-of-file is detected.

2. For nonsequential file processing, to make available a specific record from a direct-access file and to allow execution of statements if the contents of the associated symbolic and/or actual key is found to be invalid.

The format of the READ statement is:

```
READ file-name RECORD [INTO data-name]

                {AT END       }
                {INVALID KEY  }   imperative statement...
```

When a READ statement is executed, the next logical record in the named file becomes accessible in the input area defined by the associated Record Description entry. The file-name must be defined by a File Description entry in the Data Division.

The record remains available in the input area until the next READ statement (or a CLOSE statement) for that file is executed. No reference can be made by any statement in the Procedure Division to information that is not actually present in the current record. Thus, it is not permissible to refer to the nth occurrence of data that appears fewer than n times. If such a reference is made, no assumption should be made about results in the object program.

If more than one logical record is described for the file, implicit redefinition of the area exists. It is the programmer's responsibility to identify which record is present in the area at any given time.

Regardless of the method used to overlap access time with processing time, an input record is made available by a READ statement prior to execution of the next READ statement.

The INTO data-name option is equivalent to a READ statement and a MOVE statement. The data-name specified must be the name of a Working-Storage or Saved-Area record or a previously opened output record. When this option is used, the current record becomes available in the input area, as well as in the area specified by data-name. Data will be moved into the data-name area in accordance with the rules moving an item to a receiving field which is a group item.

The AT END option is required for files for which access is sequential. The AT END portion of the READ statement is executed when an end-of-file condition is detected.

Once the AT END portion of a READ statement has been executed for a file, any subsequent attempt to read from that file or to refer to logical records in that file constitutes an error, unless subsequent CLOSE and OPEN statements have been executed.

The INVALID KEY option is required for files specified as ACCESS RANDOM. The statements following INVALID KEY are executed when the contents of actual key and/or symbolic key are invalid.

If ACCESS RANDOM is specified for the file, the symbolic key and/or the actual key of the file must be set to the desired values prior to the execution of the READ statement.

Each time an end-of-volume condition occurs on a file, the READ statement causes the following operations to take place:

1. The volume trailer label checking procedure of IOCS is executed. The user trailer label checking procedures specified in a USE Option 2 sentence are executed, if such labels exist.
2. A volume switch occurs.
3. The volume header label checking procedure subroutine of IOCS is executed. The user header label checking procedures specified in a USE Option 2 sentence declarative are executed, if such labels exist.
4. The next logical record in the file is made available for processing.

If the end-of-volume is also the logical end of file, only the operations specified in item 1 are done and then the statements following AT END are executed.

WRITE

The function of the WRITE statement is to release a logical record specified as OUTPUT or I-O in an OPEN statement and to allow performance of specified imperative statements if, for random access files, the contents of the associated actual key and/or symbolic key is found to be invalid.

The format of the WRITE statement is:

```
WRITE record-name [FROM data-name-1]

    [INVALID KEY imperative statement...]

    AFTER ADVANCING {data-name-2} LINES
                    {integer    }
```

An OPEN statement must be executed prior to executing the first WRITE statement for a file. After the WRITE statement is executed, the logical record named by record-name is no longer available.

When the FROM option is used, data-name-1 must not be the name of an item in the file containing record-name. This form of the WRITE statement is equivalent to the statement MOVE data-name-1 TO record-name followed by the statement WRITE record-name. Moving takes place according to the rules specified for moving an item to a receiving field which is a group item.

After execution of a WRITE statement with the FROM option, the information in record-name is no longer available, but the information in data-name-1 is available.

When the end-of-volume condition occurs, the WRITE statement causes the following operations to take place:

1. The volume trailer label writing procedure of IOCS is executed. The user trailer label creating procedure, if specified in a USE Option 1 declarative, is executed.
2. A volume switch occurs
3. The volume header label writing procedure of IOCS is executed. The user header label writing procedure, if specified in a USE option 1 declarative, is executed.
4. The next logical record area in the output file is made available.

If ACCESS RANDOM is specified, the symbolic and/or actual key must be set to the desired values prior to the execution of the WRITE statement.

The AFTER ADVANCING option is used for output destined to be printed or punched. When this option is used, the first character in each logical record for the file must be reserved for the control character. When the AFTER ADVANCING option is used, integer must be unsigned and have the value 0, 1, 2, or 3. The value 0 designates a carriage-control 'eject' (i.e., skip to next page). The value 1 designates single spacing; the value 2, double spacing; and the value 3, triple spacing.

Data-name-2 must be an alphanumeric item of length one (i.e., must have PICTURE X). The following chart shows the values that data-name-2 may assume and their interpretations.

Value	Interpretation
b (blank)	single spacing
0	double spacing
-	triple spacing

Value	Interpretation
+	suppress spacing
1 through 9	skip to channel 1 through 9, respectively
A,B,C	skip to channel 10, 11, 12, respectively
V,W	pocket select 1 or 2, respectively on the IBM 1442, and 4 or 8 on the IBM 1402

Ext REWRITE

The function of the REWRITE statement is to replace a logical record on a direct-access device with a specified record, and to allow execution of a specified procedure if the contents of the associated actual key and/or symbolic key is found to be invalid.

The format of the REWRITE statement is:

REWRITE record-name [FROM data-name]

[INVALID KEY imperative-statement ...]

A READ statement for a file must be executed before a REWRITE statement for a file can be executed. A REWRITE statement can only be written for files opened as I-O.

When the FROM option is used, data-name must not be the name of an item in a file containing record-name. This form of the REWRITE statement is equivalent to the statement MOVE data-name TO record-name followed by the statement REWRITE record-name. Moving takes place according to the rules specified for moving an item to a receiving field which is a group item.

For sequential access files, the INVALID KEY procedure is executed when the end of file is reached.

For random access files, the INVALID KEY procedure is executed when the contents of the actual key and/or the symbolic key are not within file limits.

If ACCESS RANDOM is specified for the file, the actual key and/or the symbolic key must be set to the desired values prior to the execution of the REWRITE statement.

CLOSE

The CLOSE statement is used to terminate the processing of one or more units or files. The format of the CLOSE statement is:

```
CLOSE {file-name [UNIT] [WITH {NO REWIND}]} ...
                              {LOCK     }
```

When a CLOSE statement is specified, IOCS closing procedures are executed for the current unit of the file. The CLOSE statement may only be specified for a file that is open. After a CLOSE statement has been executed for a file, an OPEN statement must be executed before any other reference can be made to that file.

If the UNIT option is specified, the IOCS volume switching procedures are instituted.

A CLOSE statement for a file opened as I-O performs the same label checking functions as for an input file.

A CLOSE statement with the UNIT option or with the UNIT WITH LOCK option should only be written for files assigned to specific devices on which removable volumes may be mounted. The LOCK option causes the current volume of the file to be removed.

The NO REWIND option should only be written for files assigned to UTILITY device-numbers for which rewinding is possible, e.g., 2400, 7340, etc. This option suppresses rewinding normally associated with closing a file.

DISPLAY

The function of the DISPLAY statement is to display data on an output device. The format of the DISPLAY statement is:

```
          {data-name}          [UPON CONSOLE]
DISPLAY   {literal  } ...      [UPON SYSPCH ]
```

When UPON SYSPCH or UPON CONSOLE is omitted, the system logical output device (SYSOUT) is assumed. When UPON SYSPCH is written, the system logical punch device is assumed.

When UPON SYSPCH or UPON CONSOLE is written, the sum of the sizes of the operands may not exceed 72 character positions. When UPON SYSPCH and UPON CONSOLE are omitted, the sum of the sizes of the operands may not exceed the maximum logical record length for the system logical output device (SYSOUT).

Any spaces desired between displayed multiple operands must be explicitly specified.

When SYSPCH is written, an 80 character output record is produced, with positions 73 through 80 of the record containing the identification of the originating program (program-name). If the message size exceeds 72 characters, it is truncated; if less than 72, the remaining positions are filled with spaces.

Data-names described as USAGE COMPUTATIONAL, COMPUTATIONAL-1, COMPUTATIONAL-2, or COMPUTATIONAL-3 are converted automatically to external format as follows:

1. Internal decimal and binary items are converted to external decimal. Only negative values cause a low-order sign overpunch to be developed.
2. Internal floating point items are converted to external floating point. No other data items require conversion.

For example, if two binary items have values -32 and 32, then they will be displayed as 3K and 32, respectively.

ACCEPT

The function of the ACCEPT statement is to obtain data from the system logical input device (SYSIN), or from the console.

The format of the ACCEPT statement is:

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

Data-name may be either a fixed-length group item or an elementary alphabetic, alphanumeric, external decimal or external floating-point item. One logical record is read and the appropriate number of characters is moved into the area reserved for data-name. No editing or error-checking of the incoming data is done.

When FROM CONSOLE is specified data-name may not exceed 72 character positions in length.

When an ACCEPT statement with the FROM CONSOLE option is executed, the following action is taken:

1. A system-generated message code is automatically displayed followed by the literal 'AWAITING REPLY'.

2. Execution is suspended. When a console input message, preceded by the same message code as in 1 above, is identified by the Control Program, execution of the ACCEPT statement is resumed and the message is moved to the specified data-name.

The message code serves as a key by which the Control Program correlates console input with the proper program.

When the FROM CONSOLE option is not written, one logical record is read from the system logical input device (SYSIN).

Figure 14 states restrictions of input-output statements. Y means that the statement may appear; N indicates that it must not.

Statement	Appearing In:					
	Label Checking Declarative	Label Creating Declarative	Asynchronous Processing Declarative	Report Writing Declarative	Main Body of Procedure Division	Debug Packet
OPEN CLOSE	Y*	Y*	N	Y*	Y	Y
READ WRITE REWRITE	Y*	Y*	Y	Y*	Y	Y
DISPLAY	Y	Y	Y	Y	Y	Y
ACCEPT FROM CONSOLE	Y	Y	Y	Y	Y	Y
ACCEPT (from SYSIN)	Y	Y	N	Y	Y	Y

*Only permitted for files other than the one for which entry into the declarative was made.

Figure 14. Restrictions for Input/Output Statements.

ASYNCHRONOUS PROCESSING STATEMENTS F Only

PROCESS F Only

The function of the PROCESS statement is to initiate a set of out-of-line procedures associated with a direct-access file which may processed asynchronously.

The format of the PROCESS statement is:

PROCESS section-name

The PROCESS statement may only be used when asynchronous processing is used. It may only appear in the in-line portion of a program. Section-name must be the name of an out-of-line procedure written in a USE FOR RANDOM PROCESSING declarative.

The saved area and the files to be processed by the out-of-line procedure are specified in an Option 4 APPLY clause in the Environment Division.

One saved area record is automatically associated at object time with each out-of-line processing cycle. No more than one processing cycle has access to a given saved area record at any one time. The specific saved area record associated with each out-of-line processing cycle is automatically released for further storage assignment upon the completion of the processing cycle.

The processing of data in the saved area by the out-of-line procedural statements may be performed asynchronously. Thus, the in-line procedures should not refer to other data being processed in the out-of-line procedures and conversely the out-of-line procedures may not refer to any data being processed by the in-line procedures.

F Only HOLD

The function of the HOLD statement is to provide a delay point so that synchronous processing may be resumed.

The format of the HOLD statement is:

```
HOLD    [ALL                    ]
        [section-name ...]
```

A HOLD statement is meaningful only when used with asynchronous processing cycles initiated by a PROCESS statement.

A HOLD ALL statement may only be used in in-line procedures. The statement assures that all previously initiated asynchronous cycles have been completed before any statements following the HOLD ALL statement are executed.

A HOLD appearing in-line must name one or more sections or ALL. The statement assures that all previously initiated asynchronous cycles pertaining to the sections named have been have been completed before any statements following the HOLD statement are executed. The section names used as operands of the HOLD statement must be the names of sections specified in a USE FOR RANDOM PROCESSING declarative.

An out-of-line HOLD must not have any operands. It causes out-of-line processing cycles to be processed in the order in which they were initiated. There may only be one HOLD statement per out-of-line declarative procedure. The logic of USE FOR RANDOM PROCESSING declaratives must be such that every path flows through the out-of-line HOLD.

DATA MANIPULATION STATEMENTS

MOVE

The MOVE statement is used to move data from one area of main storage to another and to perform conversions and/or editing on the data that is moved. The MOVE statement has the following two formats:

Option 1

```
        {data-name-1}
MOVE    {literal    } TO data-name-2 ...
```

If Option 1 (the simple move) is used, the data represented by data-name-1 or the specified literal is moved to the area designated by data-name-2. The same information is also moved to any additional receiving areas mentioned in the statement.

When a group item is involved in a simple move, the data is moved without regard to the level structure of the group items involved and without editing.

The following considerations pertain to moving items:

1. Numeric (external decimal, internal decimal, binary, external floating, internal floating, numeric literals, and ZERO) to numeric or report:

   a. The items are aligned by decimal points, with insertion of zeros or truncation on either end, as required.

   b. When the USAGE of the source field and receiving field differs, conversion to the USAGE of the receiving field takes place.

   c. The items may have special editing performed on them with suppression of zeros, insertion of a dollar sign, commas, etc., and decimal point alignment, as specified by the receiving area.

2. All other permissible combinations:
   a. The characters are placed in the receiving area from left to right, unless the receiving field is specified as JUSTIFIED RIGHT.
   b. If the receiving field is not completely filled by the data being moved, the remaining positions are filled with spaces.
   c. If the source field is longer than the receiving field, the move is terminated as soon as the receiving field is filled.

Figure 15 contains several examples illustrating MOVE.

Source Field		Receiving Field		
PICTURE	Value	PICTURE	Value before MOVE	Value after MOVE
99V99	1234	99V99	9876	1234
99V99	1234	99V9	987	123
9V9	12	99V999	98765	01200
XXX	A2B	XXXXX	Y9X8W	A2Bbb
9V99	123	99.99	87.65	01.23
AAAAAA	REPORT	AAA	JKL	REP

Figure 15. Examples of Data Movement.

Note that,in the fourth example, the information in any excess positions of a non-numeric receiving area is replaced by spaces at the right.

Option 2 F Only

MOVE CORRESPONDING data-name-1 TO data-name-2...

When Option 2 is used, selected items within data-name-1 are moved along with any required editing, to selected areas within data-name-2. Items are selected by matching the data-names of areas defined within data-name-1 with like data-names of areas defined within data-name-2.

The rules stated for the simple MOVE apply to each pair of corresponding items in the MOVE CORRESPONDING; thus, the effect of a MOVE CORRESPONDING statement is equivalent to a series of simple MOVE statements.

The following rules apply to the CORRESPONDING option:

1. At least one of the items of a pair of matching items must be an elementary item.

2. Items are corresponding data items if the respective data-names are the same, including all qualification up to but not including data-name-1 and data-name-2.

3. Data-name-1, data-name-2, etc. must be group items.

4. There must be no OCCURS clauses governing any of the elementary items altered or moved by a MOVE CORRESPONDING statement.

5. In determining which are corresponding data- items, only the first complete description of any area will be considered in the case where a REDEFINES clause has been used. Consider the following data organization:

```
01 A
02 B
02 C REDEFINES B
03 D
03 E
02 F
```

   Only B or F can be considered as potential corresponding items. This restriction does not preclude data-name-1 or data-name-2 from having REDEFINES clauses, or from being subordinate to data-names with REDEFINES clauses.

6. Neither data-name-1 nor data-name-2 can be data items with level numbers of 77 or 88.

7. Each corresponding source item is moved in conformity with the description of the receiving area.

To illustrate the use of MOVE CORRESPONDING, suppose that the programmer wishes to transfer corresponding items from a work area named INVENTORY-POSTING to an output area designated INVENTORY-RECORD. He could write this statement:

```
MOVE CORRESPONDING INVENTORY-POSTING TO INVENTORY-RECORD
```

Figure 16 shows the movement of data that might result from this statement. Note that non-corresponding items in the source area are not moved and that non-corresponding items in the receiving area are not affected.

Figure 17 represents all permissible moves using the MOVE statement. The letter Y indicates a valid move and the letter N indicates an invalid move. A detailed description of the types of fields represented may be found in Section 5.

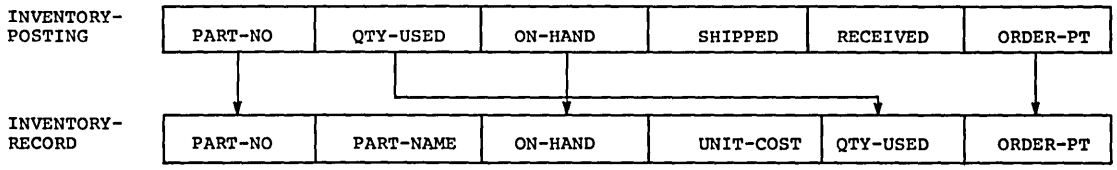


Figure 16. Movement of Data Resulting from Execution of MOVE CORRESPONDING.

	Receiving Field								
Source Field	GR	AL	AN	ED	ID	BI	EF	IF	RP
Group (GR)	Y	Y	Y	N	N	N	N	N	N
Alphabetic (AL)	Y	Y	Y	N	N	N	N	N	N
Alphanumeric (AN)	Y	Y	Y	N	N	N	N	N	N
External Decimal (ED)	Y	N	N	Y	Y	Y	Y	Y	Y
Internal Decimal (ID)	Y	N	N	Y	Y	Y	Y	Y	Y
Binary (BI)	Y	N	N	Y	Y	Y	Y	Y	Y
External Floating-Point (EF)	Y	N	N	Y	Y	Y	Y	Y	Y
Internal Floating-Point (IF)	Y	N	N	Y	Y	Y	Y	Y	Y
Report (RP)	Y	N	Y	N	N	N	N	N	N
ZEROS	Y	N	Y	Y	Y	Y	Y	Y	Y
SPACES	Y	Y	Y	N	N	N	N	N	N
ALL 'character', HIGH-VALUES, LOW-VALUES, QUOTES	Y	N	Y	N	N	N	N	N	N

Figure 17. Permissible Moves.

EXAMINE

The EXAMINE statement is used to replace certain occurrences of a given character and/or to count the number of such occurrences in a data item.

The EXAMINE statement has the following two formats:

Option 1

```
EXAMINE data-name  TALLYING {ALL       } 'character-1'
                            {LEADING   }
                            {UNTIL FIRST}

    REPLACING BY 'character-2'
```

Option 2

```
EXAMINE data-name  REPLACING {ALL       } 'character-1'
                             {LEADING   }
                             {UNTIL FIRST}
                             {FIRST     }

    BY 'character-2'
```

Data-name in each option must refer to a data item whose USAGE is DISPLAY.

Character-1 and character-2 must be single-character non-numeric literals (i.e., enclosed in quotation marks) and members of the set of allowable characters for the data item. For example, a '2' cannot replace an 'A' in an alphabetic item.

The use of figurative constants instead of character-1 or character-2 is permitted.

When Option 1 is used, a count is made at object time of the number of occurrences of the specified character in data-name, and this count replaces the value of the special binary data item TALLY, whose length is five decimal digits. TALLY may also be used as a data-name in other procedural statements.

F Only A unique TALLY exists for each cycle of each Random Processing Declarative as well as for the synchronously processed in-line procedure in a program having random processing.

The count at object time depends on which of the following three TALLYING options is employed:

1. If ALL is specified, all occurrences of character-1 in the data item are counted.

2. If LEADING is specified, the count represents the number of occurrences of character-1 prior to encountering a character other than character-1. Examination proceeds from left to right.

3. If UNTIL FIRST is specified, the count represents the number of characters other than character-1 encountered prior to the first occurrence of character-1. Examination proceeds from left to right.

When the REPLACING option is used (either in option 1 or option 2), the replacement of characters depends on which of the following four REPLACING options is employed:

1. If ALL is specified, character-2 is substituted for each occurrence of character-1.

2. If LEADING is specified, the substitution of character-2 for character-1 terminates when a character other than character-1 is encountered, or when the righthand boundary of the data item is reached. Examination proceeds from left to right.

3. If UNTIL FIRST is specified, the substitution of character-2 terminates as soon as the first character-1 is encountered, or when the righthand boundary is reached. Examination proceeds from left to right.

4. If FIRST is specified, only the first occurrence of character-1 is replaced by character-2. Examination proceeds from left to right.

Sample EXAMINE statements showing the effect of each statement on the associated data item and the TALLY are shown in Figure 18.

EXAMINE Statement	ITEM-1 Before	Data After	Resulting Value of TALLY
EXAMINE ITEM-1 TALLYING ALL '0'	101010	101010	3
EXAMINE ITEM-1 TALLYING ALL '1' REPLACING BY '0'	101010	000000	3
EXAMINE ITEM-1 REPLACING LEADING '*' BY SPACE	**7000	7000	unchanged
EXAMINE ITEM-1 REPLACING FIRST '*' BY '$'	**1.94	$*1.94	unchanged

Figure 18. Examples of Data Examination.

TRANSFORM

Ext

The TRANSFORM statement is used to alter characters according to a transformation rule. For example, it may be used to change the characters in an item to a different collating sequence.

The format of the TRANSFORM statement is:

TRANSFORM data-name-3 CHARACTERS

FROM { figurative-constant-1 / non-numeric-literal-1 / data-name-1 } TO { figurative-constant-2 / non-numeric-literal-2 / data-name-2 }

Data-name-3 must be an elementary alphabetic, alphanumeric, or report item, or a group item.

The combination of the FROM and TO options determines what the transformation rule is. These combinations are:

	TRANSFORMATION RULE
FROM figurative-constant-1 TO figurative-constant-2	All characters in data-name-3 equal to the single character figurative-constant-1 are replaced by the single character figurative-constant-2.
FROM figurative-constant-1 TO non-numeric-literal-2	All characters in data-name-3 equal to the single character figurative-constant-1 are replaced by the single character non-numeric-literal-2.
FROM figurative-constant-1 TO data-name-2	All characters in data-name-3 equal to the single character figurative-constant-1 are replaced by the single character item, data-name-2.
FROM non-numeric-literal-1 TO figurative-constant-2	All characters in data-name-3 that are equal to any character in non-numeric-literal-1 are replaced by the single character figurative-constant-2.
FROM non-numeric-literal-1 TO non-numeric-literal-2	Non-numeric-literal-1 and non-numeric-literal-2 must be equal in length or non-numeric-literal-2 must be a single character. If equal in length, any character in data-name-3 equal to a character in non-numeric-literal-1 is

	replaced by the character in the corresponding position of non-numeric-literal-2. If the length of non-numeric-literal-2 is one, all characters in data-name-3 that are equal to any character appearing in non-numeric-literal-1 are replaced by the single character given in non-numeric-literal-2.
FROM non-numeric-literal-1 TO data-name-2	Non-numeric-literal-1 and data-name-2 must be equal in length or data-name-2 must be a single-character item. If equal in length, any character in data-name-3 equal to a character in non-numeric-literal-1 is replaced by the character in the corresponding position of data-name-2. If the length of data-name-2 is one, all characters in data-name-3 that are equal to any character appearing in non-numeric-literal-1 are replaced by the single character given in data-name-2.
FROM data-name-1 TO figurative-constant-2	All characters in data-name-3 that are equal to any character in data-name-1 are replaced by the single character figurative-constant-2.
FROM data-name-1 TO non-numeric-literal-2	Data-name-1 and non-numeric-literal-2 must be of equal length or non-numeric-literal-2 must be one character. If equal in length, any character in data-name-3 equal to a character in data-name-1 is replaced by the character in the corresponding position of non-numeric-literal-2. If the length of non-numeric-literal-2 is one, all characters in data-name-3 that are equal to any character appearing in data-name-1 are replaced by the single character given in non-numeric-literal-2.
FROM data-name-1 TO data-name-2	Any character in data-name-3 equal to a character in data-name-1 is replaced by the character in the corresponding position of data-name-2. These items can be one or more characters, but must be equal in length.

The following rules pertain to the operands of the FROM and TO options:

1. The non-numeric-literals require enclosing quotation marks, as specified in the section, "Literals."

2. Data-name-1 and data-name-2 must be elementary alphabetic, or alphanumeric items, or fixed length group items less than 257 characters in length.

3. A character may not be repeated in non-numeric-literal-1 or in the

area defined by data-name-1. If a character is repeated in data-name-1 the results will be unpredictable.

4. The allowable figurative-constants are: ZERO, ZEROS, ZEROES, SPACE, SPACES, QUOTE, QUOTES, HIGH-VALUE, HIGH-VALUES, LOW-VALUE, and LOW-VALUES.

When either data-name-1 or data-name-2 appear as a determinant of the transformation rule, the user can change the transformation rule during object time.

Figure 19 contains examples of data-name-3 results, using the figurative-constant-1 to figurative-constant-2, non-numeric-literal-1 to non-numeric-literal-2, and data-name-1 to data-name-2 combinations, respectively. (The small b represents a blank.)

Data-name-3 Before	FROM	TO	Data-name-3 After
1b7bbABC	SPACE	QUOTE	1'7"ABC
1b7bbABC	'17CB'	'QRST'	QbRbbATS
1b7bbABC	b17AEC	CBA71b	BCACC71b
1234WXY89	98YXW4321	ABCDEFGHI	IHGFEDCBA

Figure 19. Examples of Data Transformation.

ARITHMETIC STATEMENTS

The following rules apply to the arithmetic statements:

1. All data-names used in arithmetic statements must represent elementary numeric data items that are defined in the Data Division of the program, except when they are the operands of the GIVING or CORRESPONDING options.

2. The maximum size of any data-name or literal is 18 decimal digits.

3. Intermediate result fields generated for the evaluation of fixed-point arithmetic expressions assure the accuracy of the result field, except where high order truncation is necessary.

4. Decimal point alignment is supplied automatically throughout computations.

The ROUNDED and SIZE ERROR options apply to all the arithmetic statements. The GIVING option applies to all arithmetic statements but COMPUTE.

GIVING Option

If the GIVING option is written, the value of the data-name that follows the word GIVING will be made equal to the calculated result of the arithmetic operation. The data-name that follows GIVING is not used in the computation and may contain editing symbols.

If the GIVING option is not written, the operand following the words TO, FROM, BY, and INTO in the ADD, SUBTRACT, MULTIPLY, and DIVIDE statements, respectively, must be a data-name. This data-name is used the computation and is made equal to the result.

ROUNDED Option

If, after decimal-point alignment, the number of places in the calculated result is greater than the number of places associated with the data-name whose value is to be set equal to the calculated result, truncation occurs unless the ROUNDED option has been specified.

When the ROUNDED option is specified, the least significant digit of the resultant data-name has its value increased by 1 whenever the most significant digit of the excess is greater than or equal to 5.

Rounding of a computed negative result is performed by rounding the absolute value of the computed result and then making the final result negative (unless the final result is zero).

Figure 20 illustrates the relationship between a calculated result and the value stored in an item that is to receive the calculated result.

	Item to	Receive Calculated Result	
Calculated Result	PICTURE	Value After Rounding	Value After Truncating
12.36	99V9	12.4	12.3
8.432	9V9	8.4	8.4
35.6	99V9	35.6	35.6
65.6	99V	66	65
.0055	V999	.006	.005

Figure 20. Rounding or Truncation of Calculations.

SIZE ERROR Option

Whenever the number of integral places in the calculated result exceeds the number of integral places specified for the resultant data-name, a size error condition arises.

If the SIZE ERROR option has been specified and a size error condition arises, the value of the resultant data-name is not altered and the series of imperative statements specified for the condition is executed.

If the SIZE ERROR option has not been specified and a size error condition arises, no assumption should be made about the final result; but the program flow is not interrupted.

It should be noted that the SIZE ERROR option applies only to final calculated results. When a size error occurs in the handling of intermediate results, no assumption should be made about the final result.

An arithmetic statement, if written with a SIZE ERROR option, is not an imperative statement. Rather, it is a conditional statement and is prohibited in contexts where only imperative statements are allowed.

COMPUTE

The COMPUTE statement assigns to a data item the value of a numeric data item, literal, or arithmetic expression. The format of a COMPUTE statement is:

```
COMPUTE data-name-1 [ ROUNDED ] = { data-name-2            }
                                  { numeric-literal        }
                                  { floating-point-literal }
                                  { arithmetic-expression  }

    [ON SIZE ERROR imperative statement...]
```

The data-name specified to the left of the equal sign must be an elementary report, binary, internal decimal, external decimal, internal floating-point, or external floating-point item.

The ON SIZE ERROR option applies only to the final result and not to any of the intermediate results.

ADD

The ADD statement adds two or more numeric values and substitutes the resulting sum for the current value of an item. The ADD statement has the following formats:

Option 1

```
ADD { numeric-literal        }...{ TO     }
    { floating-point-literal }   { GIVING } data-name-n
    { data-name-1            }

    [ROUNDED] [ON SIZE ERROR imperative-statement...]
```

When the TO option is used, the values of all the data-names (including data-name-n) and literals in the statement are added, and the resulting sum replaces the value of data-name-n. At least two data-names and/or numeric literals must follow the word ADD when the GIVING option is written.

Option 2 F Only

```
ADD CORRESPONDING data-name-1 TO data-name-2

    [ROUNDED] [ON SIZE ERROR imperative statement...]
```

The CORRESPONDING option of the ADD statement allows the programmer to specify the addition of corresponding data items in one operation similar to MOVE statement with a CORRESPONDING option.

Numeric elementary items within data-name-1 are added to numeric elementary items with matching data-names within data-name-2. The data-names match if they and all their qualifiers up to, but not including, data-name-1 and data-name-2 are the same. There must be no OCCURS clauses governing any of the operands involved in an ADD CORRESPONDING statement.

Data-name-1 and data-name-2 must be group items. The rules stated for arithmetic statements apply to each pair of items in the ADD CORRESPONDING option.

When ON SIZE ERROR is used in conjunction with CORRESPONDING, the size error test is made only after the completion of all the ADD operations. If any of the additions produced a size error condition, the resultant field for that ADD remains unchanged, and the procedure specified in the SIZE ERROR clause is executed.

When the ROUNDED option is used in conjunction with CORRESPONDING, it applies to all the add operations.

SUBTRACT

The SUBTRACT statement subtracts one or a sum of two or more numeric data items from a specified item and sets the value of a data item equal to the difference.

The SUBTRACT statement has the following formats:

Option 1

```
        ⎧data-name-1              ⎫
SUBTRACT⎨numeric-literal-1        ⎬ ...
        ⎩floating-point-literal-1 ⎭

     ⎧data-name-m [GIVING data-name-n]                  ⎫
FROM ⎨numeric-literal-m GIVING data-name-n              ⎬
     ⎩floating-point-literal-m GIVING data-name-n       ⎭

    [ROUNDED] [ON SIZE ERROR imperative statement...]
```

The effect of the SUBTRACT statement is to add the values of all the operands that precede FROM and then to subtract the sum from the value of the item following FROM. A literal can follow FROM only when the GIVING option is specified.

F Only Option 2

```
SUBTRACT CORRESPONDING data-name-1 FROM data-name-2

    [ROUNDED] [ON SIZE ERROR imperative statement...]
```

The CORRESPONDING option of the SUBTRACT statement is analogous to the CORRESPONDING option of the ADD statement.

MULTIPLY

The MULTIPLY statement multiplies two numeric data items and sets the value of a data item equal to the product.

The format of the MULTIPLY statement is:

```
        ⎧data-name-1              ⎫
MULTIPLY⎨numeric-literal-1        ⎬
        ⎩floating-point-literal-1 ⎭

   ⎧data-name-2 [GIVING data-name-3]                ⎫
BY ⎨numeric-literal-2 GIVING data-name-3            ⎬
   ⎩floating-point-literal-2 GIVING data-name-3     ⎭

  [ROUNDED] [ON SIZE ERROR imperative statement...]
```

DIVIDE

The DIVIDE statement divides one numeric data item into another and sets the value of an item equal to the quotient.

The format of a DIVIDE statement is:

```
        ( data-name-1                 )
DIVIDE  ( numeric-literal-1           )
        ( floating-point-literal-1    )

INTO    ( data-name-2 [GIVING data-name-3]                )
        ( numeric-literal-2 GIVING data-name-3            )
        ( floating-point-literal-2 GIVING data-name-3     )

[ROUNDED] [ON SIZE ERROR imperative statement...]
```

Division by zero results in a SIZE ERROR condition.

PROCEDURE BRANCHING STATEMENTS

In the GO TO, ALTER, and PERFORM statements, procedure-name signifies paragraph-name or section-name.

STOP

The STOP statement is used to terminate or delay execution of the object program. The format of this statement is:

```
STOP ( RUN     )
     ( literal )
```

The STOP RUN statement terminates execution of the object program and returns control to the operating system.

The STOP literal statement causes a compiler-generated message code, and the specified literal, to be displayed and causes the object program to pause. The program may be resumed only by operator intervention. The message code must be keyed in on the console in order to resume execution.

GO TO

The GO TO statement transfers control from one portion of the program to another. The GO TO statement has the following formats:

Option 1

```
GO TO [procedure-name]
```

Option 1 of the GO TO statement provides a means of transferring the path of flow of a program to a designated paragraph or section.

Option 2

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

When Option 1 (unconditional GO TO) is used and a procedure-name is not specified, the GO TO statement must have a paragraph-name, be the only statement in the paragraph, and be modified by an ALTER statement prior to the first execution of the GO TO statement. The paragraph-name assigned to the GO TO statement is referred to by the ALTER statement in order to modify the sequence of the program. If procedure-name is omitted and the GO TO statement has not been preset by an ALTER statement prior to the first execution of the GO TO statement, execution of the program is terminated.

In Option 2, data-name must be an elementary integral numeric item whose length does not exceed four digits and whose usage is either DISPLAY, COMPUTATIONAL, or COMPUTATIONAL-3.

Option 2 specifies alternative branch points; control is transferred to the point specified by the value of data-name. Control goes to the 1st, 2nd,...,nth procedure-name as the value of data-name is 1, 2,...,n. If data-name has a value outside the range 1 to n, no transfer takes place, and control passes to the next statement after the GO TO statement.

ALTER

The ALTER statement is used to modify an unconditional GO TO statement elsewhere in the Procedure Division, thus changing the sequence in which program steps are to be executed.

The format of the ALTER statement is:

ALTER {procedure-name-1 TO PROCEED TO procedure-name-2}...

Procedure-name-1 designates a paragraph containing a single sentence consisting only of an Option 1 GO TO statement. The effect of an ALTER statement is to replace the procedure-name specified in Option 1 of the GO TO statement with procedure-name-2 of the ALTER statement, where the paragraph-name containing the GO TO statement is procedure-name-1 in the ALTER statement.

PERFORM

The PERFORM statement specifies a transfer of control from one portion of a program to another, in order to execute some procedure a specified number of times, or until a condition is satisfied. It directs that control is to be returned to the statement immediately following the point from which the transfer was made.

The PERFORM statement has the following four formats:

Option 1

PERFORM procedure-name-1 [THRU procedure-name-2]

Option 1 is the simple PERFORM statement. A procedure referred to by this type of PERFORM statement is executed once,and then control passes to the next statement after the PERFORM statement. All statements in the paragraphs or sections named by procedure-name-1 through procedure-name-2 constitute the range of the PERFORM statement.

Option 2

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

    {integer  }
    {data-name} TIMES
```

Option 2 is the TIMES option. When the TIMES option is used, the procedure is performed the number of times specified by data-name or integer. Control is then transferred to the statement following the PERFORM statement. Data-name must have an integral value and data-name or integer must have a positive value, less than 32,768. If the value of the data-name is negative, zero, or greater than 32,767, control is passed immediately to the statement following the PERFORM statement.

Option 3

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

    UNTIL test-condition
```

Option 3 is the UNTIL option. Test-condition may be simple or compound. The procedures specified by the procedure-names are performed until the condition specified by the UNTIL option is true. At this time, control is transferred to the statement following the PERFORM statement. If the condition specified by the UNTIL option is true at the time the PERFORM statement is encountered, the specified procedure is not executed.

Option 4

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

  VARYING data-name-2     FROM {numeric-literal-2}
                               {data-name-2      }

  BY {numeric-literal-3}  UNTIL test-condition-1
     {data-name-3      }

 [AFTER data-name-4]      FROM {numeric-literal-4}
                               {data-name-5      }

  BY {numeric-literal-6}  [UNTIL test-condition-2]
     {data-name-6      }

 [AFTER data-name-7]      FROM {numeric-literal-8}
                               {data-name-8      }

  BY {numeric-literal-9}  [UNTIL test-condition-3]
     {data-name-9      }
```

Option 4 is the VARYING option. Test-condition may be simple or compound.

This option is used to augment the value of one or more data-names in an orderly fashion during the execution of a PERFORM statement. When one data-name is varied, data-name-1 is set equal to its starting value (FROM) when commencing the PERFORM statement. Then, test-condition-1 is evaluated: if it is true, control passes to the next statement following

the PERFORM statement; if false, procedure-name-1 through procedure-name-2 is executed once. The value of the decrement (BY), and the condition (UNTIL) is evaluated again. The cycle continues until test-condition-1 is true, at which point control is passed to the statement following the PERFORM statement.

When two data-names are varied, the value of data-name-4 goes through a complete cycle (FROM, BY, UNTIL) each time that data-name-1 is augmented with its BY value. For three data-names, the value of data-name-7 goes through a complete cycle (FROM, BY, UNTIL) each time that data-name-4 is augmented with its BY value, which in turn goes through a complete cycle each time data-name-1 is varied.

Regardless of the number of data-names being varied, as soon as test-condition-1 is found to be true, control is transferred to the next statement after the PERFORM statement.

All data-names and literals used in Option 4 must represent numeric values, and need not be integers; they may be positive, negative, or zero. If compatibility with previous COBOL compilers is desired and the AFTER option is used, all data-names and literals must represent integral values. Data-name-1, data-name-4, and data-name-7 must not be alternate names for the same data items. For all options, the first statement of procedure-name-1 is the point to which sequence control is transferred by the PERFORM statement.

The return of control is from a point determined as follows:

1. If procedure-name-1 is a paragraph-name and procedure-name-2 is not specified, the return is made after the last statement of the procedure-name-1 paragraph.

2. If procedure-name-1 is a section-name and procedure-name-2 is not specified, the return is made after the last statement of the last paragraph of the procedure-name-1 section.

3. If procedure-name-2 is specified and is a paragraph-name, the return is made after the last statement of the procedure-name-2 paragraph.

4. If procedure-name-2 is specified and is a section-name, the return is made after the last statement of the last paragraph of the procedure-name-2 section.

GO TO statements and other PERFORM statements are permitted between procedure-name-1 and the last statement of procedure-name-2. Furthermore, the time sequence of execution of exits established by PERFORM statements must be in the inverse order in which they were established.

The exact range of a PERFORM statement must not be activated again while the range is currently active. An active PERFORM statement, whose execution point begins within the range of another PERFORM, must not contain the exit point of the other active PERFORM. If the logic of a procedure requires a conditional exit prior to the final sentence, the EXIT sentence must be used. In this case, procedure-name-2 must be the name of the paragraph that consists solely of the EXIT sentence.

A procedure referred to by one PERFORM statement can be referred to by other PERFORM statements. Moreover, a procedure referred to by one or more PERFORM statements can also be executed by "dropping through," that is, by entering the procedure through the normal passage of control from one statement to the next, in sequence. Accordingly, if procedure-name-1 were the next statement following the PERFORM statement, the procedure would be executed one time more than specified

by the PERFORM statement because, after execution of the PERFORM statement, control would pass to procedure-name-1 in the normal continuation of the sequence.

At any time during the execution of an object program, no paragraph-name may be the terminus of the range of more than one active PERFORM statement being executed.

Figures 21, 22, and 23 illustrate the logical flow of Option 4 PERFORM statements, varying one, two, and three data-names, respectively.

Figure 24 states restrictions on the appearance of procedure-branching statements. Y means that the statement may appear; N indicates that it must not; text indicates the outcome if the statement does appear.

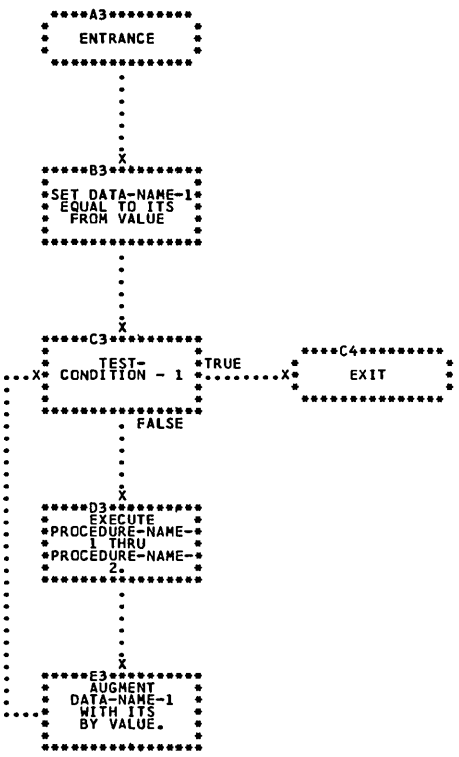


Figure 21. Logical Flow of Option 4 PERFORM Statement Varying One Data-name.

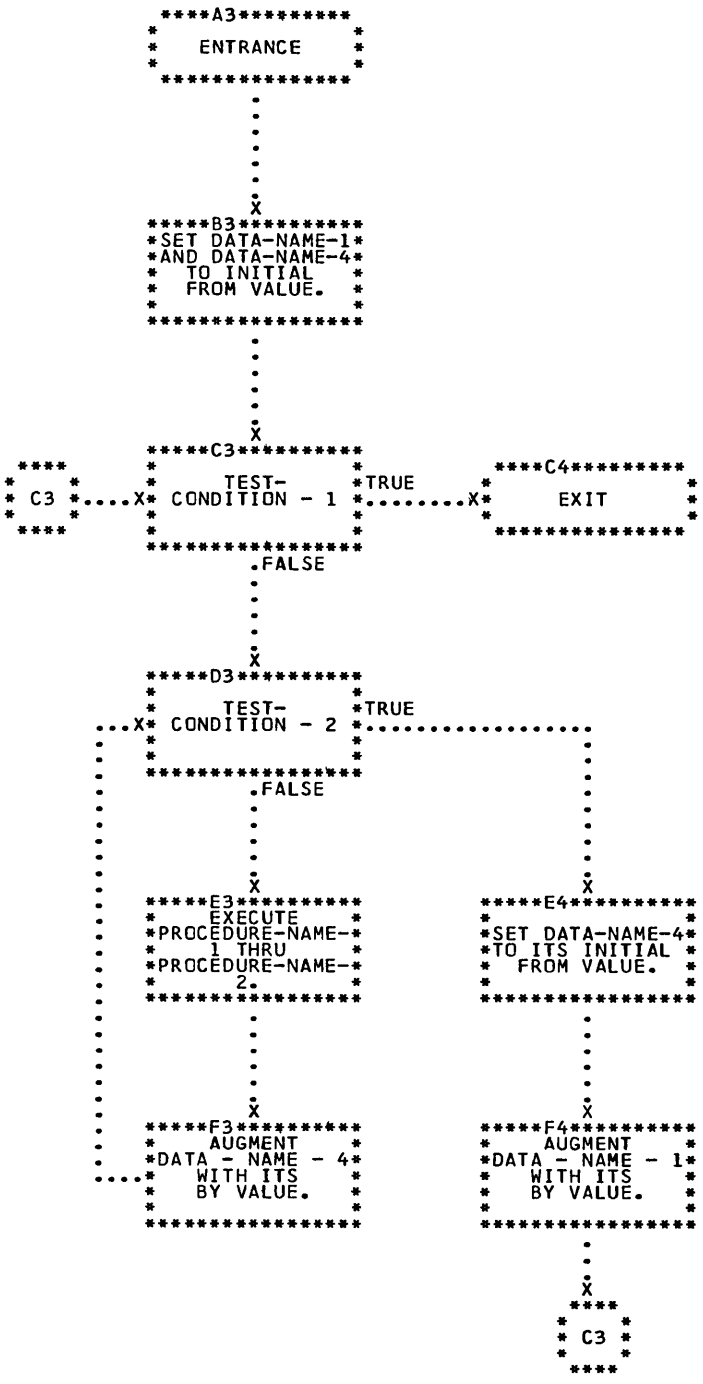


Figure 22. Logical Flow of Option 4 PERFORM Statement Varying Two Data-names.

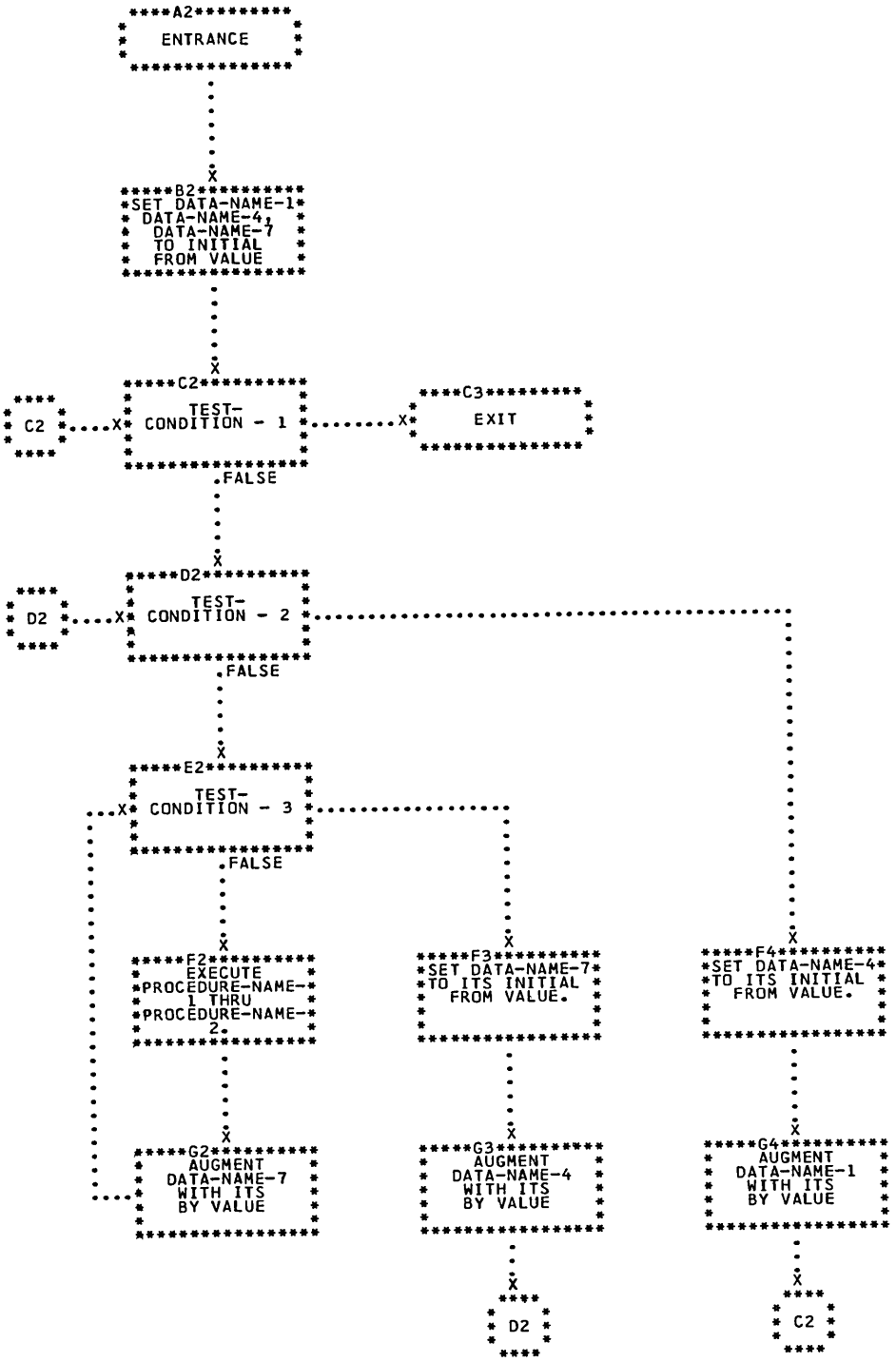


Figure 23. Logical Flow of Option 4 PERFORM Statement Varying Three Data-names.

Statement	Appearing In:					
	Label Checking Declarative	Label Creating Declarative	Asynchronous Processing Declarative	Report Writing Declarative	Main Body of Procedure Division	Debug Packet
GO TO PERFORM ALTER	Y*	Y*	Y*	Y*	Y**	Y***
STOP RUN	N	N	N	N	end of execution	abnormal end of execution
STOP literal	Y	Y	Y	Y	Y	Y

*Operands of these statements must be procedure-names appearing in the declarative containing the statement.

**Operand of these statements must be procedure-names appearing in the main body of the Procedure Division.

***Operands of these statements may be procedure-names appearing either in the main body or in any debug packet.

Figure 24. Restrictions for Procedure-Branching Statements.

COMPILER-DIRECTING STATEMENTS

Compiler-directing statements must be separate sentences.

ENTER

The ENTER statement, used in conjunction with CALL or ENTRY statements, permits communication between a COBOL object program and one or more COBOL subprograms or other language subprograms.

The ENTER statement has the following two formats:

Option 1 (Used in calling program)

```
ENTER LINKAGE.
CALL entry-name [USING argument...].
```

Option 2 (Used in a COBOL subprogram)

```
ENTER LINKAGE.
ENTRY entry-name [USING data-name...].
ENTER COBOL.

subprogram statements

ENTER LINKAGE.
RETURN [VIA entry-name].
ENTER COBOL.
```

Entry-name is an external name and must follow the rules for external name formation.

Option 1 is used to effect transfer of control to a subprogram. Entry-name represents the name of the subprogram's entry point.

In the USING option, an argument may be one of the following:

1. A data-name when calling a COBOL subprogram
2. A data-name, file-name, or a procedure-name when calling a subprogram written in a language other than COBOL.

Option 2 is used to establish an entry point in a COBOL subprogram. Control is transferred to the entry point by a CALL statement in another program. Entry-name defines the entry point where link parameters are saved for eventual return and address parameters are obtained.

Each data-name in the USING portion of the ENTRY statement must be defined in the Linkage Section of the Data Division, and must have level number 01 or 77.

Computer base addresses of data items named in the USING list of an ENTRY statement are obtained from the USING list of the associated CALL statement. Names in the two USING lists (that of the CALL in the main program, and that of the ENTRY in the subprogram) are paired in one-to-one correspondence.

There is no necessary relationship between the actual names used for such paired names, but the data descriptions must be equivalent. When a group data item is named in the USING list of an ENTRY statement, names subordinate to it in the subprogram's Linkage Section may be employed in subsequent subprogram procedural statements, when elementary items in the group are utilized.

RETURN VIA entry-name enables restoration of the necessary registers saved at an entry point. The return from a subprogram is always to the first instruction after the calling sequence of the main program.

There must be no path of program flow into an Option 2 ENTER statement within the program containing the ENTER statement. Hence, the statement should not have a paragraph-name.

EXIT

The EXIT statement may be used when it is necessary to provide an end point for a procedure that is to be executed by means of a PERFORM statement or for a procedure that is a declarative.

The format for the EXIT statement is:

paragraph-name. EXIT.

EXIT must appear in the source program as a one-word paragraph preceded by a paragraph-name.

When the PERFORM statement is used, an EXIT paragraph-name may be the procedure-name given as the object of the THRU option. In this case, a statement in the range of a PERFORM being executed may transfer to an EXIT paragraph, bypassing the remainder of the statements in the PERFORM range. In all other cases, EXIT paragraphs have no function and control passes sequentially through them to the first sentence of the next paragraph.

NOTE

The NOTE statement permits the programmer to write explanatory comments, in the Procedure Division of a source program, which will be produced on the listing but serve no other purpose. The format of the NOTE statement is:

NOTE comment... .

NOTE, when used, must begin a sentence. Following the word NOTE, any combination of the characters from the COBOL character set may appear. If NOTE is the first word of a paragraph, any remaining sentences within the paragraph are also considered notes. Proper format rules for paragraph structure must be observed.

The Report Writer feature permits the programmer to specify the format of his output, when that output is to be a printed report.

Previously, if a COBOL programmer wanted his output in report format, he had to: specify that his headings, footings, and individual items be printed each time they were to appear; keep track of the line count and page count; accumulate totals; and provide for "overflow" situations, where a page ended before an entire group of data had been printed.

When the Report Writer feature is used, headings and footings are described once in the Data Division and are printed automatically at object time; line counters and page counters are incremented automatically; new pages are begun and page or overflow headings and footings are printed automatically; data to be accumulated is summed and the totals are printed automatically. In other words, after the programmer has manipulated his input data, the Report Writer puts the data on tape (or other intermediate storage media), in the specified format, for off-line printing. It is also possible to print a report on-line.

Another useful aspect of the Report Writer feature is that several reports can be generated within one program. A single-character identification code can be added to each line of output to specify the report to which it belongs. In this way, the output of several reports can be interspersed on one output file and can then be selected by the identification code for off-line printing.

The major entries of the Report Writer feature are made in the Data Division; therefore, the programmer must be familiar with the material covered in Section 5. When the Report Writer feature is used, an entry is required in the File Section to list the names of the reports to be produced, and a Report Section is required at the end of the Data Division to define these reports.

The Report Section contains a format description of each report. The entries in this section stipulate:

1. The maximum number of lines that can appear on a page.
2. The content of the headings and footings, when they are required, and where they are to appear on the page.
3. The format of the data, the source of the data, and where it is to appear on the page.
4. The data to be summed, and when and where the totals are to appear on the page.

In a program that utilizes the Report Writer feature, records are read and data is manipulated prior to entering the report phase. A report is produced by the execution of the INITIATE, GENERATE, and TERMINATE statements in the Procedure Division. The INITIATE statement is used to initialize all counters associated with the Report Writer, the GENERATE statement is used each time a detailed portion of the report is to be produced, and the TERMINATE statement is used to end the report. The Report Writer feature allows additional manipulation of data by means of a USE BEFORE REPORTING declarative in the Declaratives portion of the Procedure Division.

ILLUSTRATIONS

Figure RW-1 and RW-2 illustrate the report concept, and the items discussed in the following text are keyed within the figures. Figure RW-1 is a listing of a COBOL source program that uses the Report Writer feature. Figure RW-2 is the report produced by the program shown in Figure RW-1.

DATA DIVISION CONSIDERATIONS

When the Report Writer feature is used, entries must be made in the Data Division to describe the format and content of the report. An entry must be made in the File Section to list the names of the reports to be produced by the Report Writer, and a Report Section must be included at the end of the Data Division.

FILE SECTION REPORT CLAUSE

The name of each report to be produced by the Report Writer must appear in a REPORTS clause in the File Section in a File Description entry for an output file. The format of the clause is:

```
{REPORT IS  }
{REPORTS ARE}      report-name ...
```

Figure RW-1 (1) shows this clause used in a File Description entry. The presence of two or more report-names in this clause indicates that the file contains more than one report. These reports may be of different sizes, formats, etc., and the order in which their names appear in the clause is not significant.

A report may be incorporated in more than one file. In this case, the report-name must appear in a REPORT Clause of the File Description entry for each file that contains the report.

The report or reports are printed on-line by the object program when:

a. A REPORTS clause is written in a File Description entry having no 01 level record descriptions subordinate to it.

b. The file described with the REPORTS clause and named in the SELECT sentence of the Environment Division is a printer assigned to UNIT-RECORD.

REPORT SECTION

The Report Section must begin with the header REPORT SECTION, followed by a period (line 01490 of Figure RW-1).

The Report Section contains specifications for the physical layout of each page of a report and reflects the user's logical organization of the report. It consists of three distinct types of entries which are described in the following text: Report Description entries, Report Group Description entries, and Report Element Description entries.

REPORT STRUCTURE

The structure of a report is specified by the three types of Report Section entries. RD is the level indicator for a Report Description entry. Such an entry specifies clauses which apply to the entire report (Figure RW-1 (3)).

Each report is composed of items called report groups. A report group is a unit of related data, regardless of its physical size or layout. This unit can be a detail line (Figure RW-2 (10)), a set of report headings (Figure RW-2 (2)), or a set of report footings (Figure RW-2 (8) (7) (9)). A report group may extend over several actual lines of a page (Figure RW-2 (1) (2)). There are four types of headings and footings: control, page, overflow, and report. Examples of headings are shown in Figure RW-2 (1) (2) (4), examples of footings are shown in Figure RW-2 (5) (6) (7) (8) (9), and an example of a detail line is shown in Figure RW-2 (10).

The type of report group and its location on a page is specified by the clauses of a Report Group Description entry (Figure RW-1 (4)). Such an entry has the level number 01. A detailed, element by element description of the items that compose each report group is specified by the clauses of a Report Element Description entry (Figure RW-1 (5)). Such an entry may have the level number 02 through 49 and must appear immediately after the corresponding Report Group Description entry.

From the preceding discussion, it can be seen that the level structure employed in the Report Section resembles that used for File and Record Description entries in the File and Working-Storage Sections, but with important differences. Record Description entries, no matter what their level indicator, are composed of the same set of clauses, with certain minor restrictions. In the Report Section, a Report Group Description entry is composed of one set of specific clauses, while the Report Element Description entry is composed of a different set of specific clauses. Unlike the level numbers 01-49 for a Record Description entry, the level number 01 and the level numbers 02-49, when used in the Report Section, identify two different types of entry. The level numbers 02-49 in the Report Section, however, resemble the level numbers 02-49 of a Record Description in that an entry with a higher level indicator is subordinate to an entry with a lower level indicator.

CONTROL BREAKS

When a change in the value of an item causes Report Writer to perform functions other than its normal operations (that is, other than the processing of Detail report groups), a control break is said to occur. Such functions as the summation of data and the resetting of sum counters may be performed when a control break occurs.

The data-names associated with values that initiate control breaks are called controls and are specified in the CONTROL clause of the Report Description entry.

In example RW-2, DAY and MONTH are controls. Each time the item DAY changes causing a control break, a summary line indicating the purchases for the day is printed and a heading line for the new day follows. Then, the Detail report group for the new day is printed. The same is true when the item MONTH changes, causing another control break.

A Report Description entry names a report and can specify an identifying code for the report, the controls for a report, and a description of the pages in a report.

Each report whose name appears in an output File Description entry must be defined by a Report Description entry. All of the data-names used in this entry must be defined in the File, Working-Storage, or Linkage Section of the Data Division. The format of the Report Description entry is:

RD report-name [CODE 'character'] [CONTROL-clause]

[PAGE-clause].

RD is the level indicator and is required.

Report-name is the name of the report to be printed. The report-name must be unique and is required.

The CODE, CONTROL, and PAGE clauses are optional, and are described in the following text.

CODE Clause

The CODE clause is used when more than one report is to be generated and the output from each report is to be stored on one intermediate device for later printing. If an object program produces an output file that contains interspersed records from several reports, the individual reports can be separated by means of the identification code. The format of the CODE clause is:

CODE 'character'

Character is a unique character that is attached, at object time, to each logical record in the report. It must be a single alphanumeric character enclosed by quotation marks, and may not be a quotation mark.

The following example shows how this clause can be coded:

```
RD EXPENSE-REPORT CODE 'X'
```

After the object program has been run and the logical records are on one output file, a print-selection program can be written to inspect each record on the file, printing only those records that are identified by X.

Note that it is possible to write such a print-selection program in COBOL.

CONTROL Clause

The CONTROL clause specifies the controls in a report and the level of the corresponding control breaks (i.e., major, intermediate, and minor). This clause is required when CONTROL FOOTING or CONTROL HEADING is specified since such footings and headings are printed only as a

result of control breaks. The format of the CONTROL clause is:

```
[{CONTROL IS  } {FINAL                  }]
 {CONTROLS ARE} {data-name...           }
                {FINAL data-name...     }
```

FINAL is the highest control for a report. If a heading is to be printed as a result of such a control, it must be classified in a TYPE clause in a Report Group Description entry. It will be printed only the first time that a GENERATE statement for a report is executed. If a footing is to be printed as the result of such a control, it must also be classified in a TYPE clause. If will be printed when a TERMINATE statement for the report is executed.

The data-names are controls. These data-names must be fixed-length items. The values of these items determine when control breaks occur in the report. The value of the item specified by the first data-name determines that point in the report when a major control break occurs. Any other names specified represent the intermediate controls (and associated control breaks); the last data-name represents the minor control. All of the data-names used in this entry must be defined in the File, Working-Storage, or Linkage Section of the Data Division. If a heading or a footing is to be printed as the result of a control break, it must be classified in a TYPE clause of a Report Group Description entry.

The following example shows how this clause can be coded:

```
RD EXPENSE-REPORT CONTROLS ARE FINAL, MONTH, DAY.
```

A control break will occur when any of the values in MONTH or DAY, changes. Line 01500 of Figure RW-1 shows how this coding is incorporated in a typical source program.

PAGE Clause

The PAGE clause is used to specify the maximum number of lines in a page and the rules for positioning report groups within the page. The PAGE clause is required when page headings and footings and/or overflow headings and footings are specified. The format of the PAGE clause is:

```
PAGE integer-p [LINES] [HEADING] integer-h]

[FIRST DETAIL integer-d]

[LAST DETAIL integer-e] [FOOTING integer-f]
```

(In the following text, p, h, d, e, and f refer to the integers specified above.)

The maximum number of lines to be printed on a page is p LINES.

h is the number of the first line on which a heading can appear. No report group can start before line h.

d is the number of the first line on which a detail report group can appear. No detail or control report group can start before line d. If headings extend beyond line d, the first detail or control report group follows the last heading line.

e is the number of the last line on which a detail report group can appear. No detail or control heading report group extends beyond line e.

f is the number of the last line on which a control footing group can appear. No control footings can start before line d or extend beyond f. Page and overflow footings follow line f, but do not extend beyond line p.

The PAGE clause entries can be summed up as follows:

h must be greater than or equal to one (1).

d must be greater than or equal to h.

e must be greater than or equal to d.

f must be greater than or equal to e.

p must be greater than or equal to f.

If the PAGE clause is specified, and one or more of the page specifications (h, d, e, f) is missing, the following assumptions are implied:

1. If h is not specified, it is assigned the value one (1).
2. If d is not specified, it is assigned the value h.
3. If e is not specified, it is assigned the value f, if f is specified.
4. If f is not specified, it is assigned the value e. If neither e nor f is specified, e and f are set equal to p.

The following example shows how this clause can be coded:

```
RD EXPENSE-REPORT CODE 'X'.
     PAGE 59 LINES HEADING 1 FIRST DETAIL 10
     LAST DETAIL 48 FOOTING 52.
```

Lines 01500 through 01520 of Figure RW-1 show how this coding is incorporated in a typical source program.

REPORT GROUP DESCRIPTION ENTRY

This entry is used to describe the characteristics of each report group within a report. The order in which report groups are described is not significant, because their position in the report is defined by the TYPE clause. The format of a Report Group Description entry is:

01 [data-name] [LINE-clause] [NEXT GROUP-clause] [TYPE-clause].

01 is the level number and is required. The level number 01 identifies the entry as a Report Group Description entry.

Data-name is the name of the report group being described. A data-name need not always be specified in an entry; if it is, it must follow the level number. A data-name must be specified when:

1. A report group is referred to by a GENERATE statement in the Procedure Division (lines 02170 and 01740 in Figure RW-1).

2. A report group is referred to by a USE BEFORE REPORTING declarative in the Declaratives Section of the Procedure Division.

3. A sum counter is referred to in the Procedure Division or in another report group. A sum counter is a compiler-generated core storage area into which values are accumulated at object time as the result of a SUM clause.

LINE, NEXT GROUP and TYPE are described in the following text. LINE and NEXT GROUP are optional clauses, and TYPE is a required clause.

LINE Clause

The LINE clause indicates the absolute or relative line number on which the report group is to be printed. When the programmer wants data to appear on a specific line, he uses absolute line numbering. If absolute line numbering is indicated in all of the LINE clauses of the Report Group Description entries, the PAGE clause need not be used. If the programmer does not want data to appear on a specific line, but wants the data to appear one or more lines beyond the previous data, he uses relative line numbering. The format of the LINE clause is:

```
[LINE{integer-1          }]
     {PLUS integer-2     }
     {NEXT PAGE          }
```

Integer-1 is an absolute line number. It must be within the range specified by the PAGE clause. At object time, if integer-1 is not greater than the previously specified value of the line counter, a page or overflow condition will exist and the report group will be printed on the next page. If integer-1 is specified, the line counter is set to the value indicated by integer-1. Page and overflow conditions and the line counters are discussed later in this text.

Integer-2 is a relative line number. At object time, if integer-2 is specified, the value indicated by integer-2 is added to the current value of the line counter.

NEXT PAGE can only appear at the 01 level. It specifies that the current report group is to be the first item that appears on the following page. The appropriate page or overflow footings and headings are produced before this report group is printed.

The LINE clause must be used in either the Report Group Description entry or the first Report Element Description entry.

If a LINE clause is specified for an item, subsequent entries will be printed on the same line unless a new LINE clause is specified.

Within a report group, entries must be assigned line numbers in ascending order.
The following example shows how this clause can be coded:

```
01  DETAIL-LINE LINE PLUS 1 TYPE DETAIL.
```

Line 01740 of Figure RW-1 shows how this coding is incorporated in a typical source program.

NEXT GROUP Clause

The NEXT GROUP clause may be used to specify the number of print lines to be skipped after the current report group. The format of this clause is:

```
[NEXT GROUP  {integer-1          }]
             {PLUS integer-2     }
             {NEXT PAGE          }
```

Integer-1 is an absolute line number and must be a positive integer. At object time, the value specified by integer-1 is placed in the line counter after the current group is printed. Integer-1 must be within the range specified by the PAGE clause. If integer-1 is not greater than the current value of the line counter, a page or overflow condition exists.

Integer-2 is a relative line number and must be a positive integer. If integer-2 is specified, the value indicated in integer-2 is added to the line counter after the current report group is printed.

NEXT PAGE specifies that the current report group is to be the last item printed on the page. The appropriate page or overflow footings and headings are produced whenever NEXT PAGE is executed.

The following example shows how this clause can be coded:

```
01 NEXT GROUP PLUS 1 TYPE OVERFLOW HEADING
```

Line 01660 of Figure RW-1 shows how this coding is incorporated in a typical source program.

TYPE Clause

The TYPE clause is required. It is used to specify the kind of report group that is being described. A report group can be one of the following nine types (when coding a program, the programmer can abbreviate these report group types as shown):

REPORT HEADING	RH	CONTROL FOOTING	CF
PAGE HEADING	PH	OVERFLOW FOOTING	OV
OVERFLOW HEADING	OH	PAGE FOOTING	PF
CONTROL HEADING	CH	REPORT FOOTING	RF
DETAIL	DE		

The format of the TYPE clause is as follows:

```
                 {REPORT HEADING                                }
                 {PAGE HEADING                                  }
                 {OVERFLOW HEADING                              }
                 {                 {data-name-1}                }
                 {CONTROL HEADING  {FINAL      }                }
TYPE             {                                              }
                 {DETAIL                                        }
                 {                 {data-name-2}                }
                 {CONTROL FOOTING  {FINAL      }                }
                 {OVERFLOW FOOTING                              }
                 {PAGE FOOTING                                  }
                 {REPORT FOOTING                                }
```

REPORT HEADING (RH) is a report group that is printed only once, at the beginning of the report. There may be only one report group of this type in a report (see line 01530 of Figure RW-1).

PAGE HEADING (PH) is a report group that is printed at the beginning of each page, following a page condition. There may be only one report group of this type in a report (see line 01580 of Figure RW-1). If PH is specified, the PAGE clause must appear in the Report Description entry containing this Report Group Description entry.

OVERFLOW HEADING (OH) is a report group that is printed at the beginning of the next page, following an overflow condition. There may be only one report group of this type in a report (see line 01660 of Figure RW-1). If OH is specified, then LAST DETAIL integer-e must be specified in the PAGE clause for this report. Overflow conditions are discussed later in this text.

CONTROL HEADING (CH) has two options:

CH data-name-1 is a report group that is printed following the control break specified by data-name-1 or by a higher-level control. There may be only one report group of this type for each control data-name.

CH FINAL is a report group that is printed only once, between the report heading and the first control heading group. There may be only one report group of this type in a report.

If this type of report group is used, data-name-1 and/or FINAL must be cited in the CONTROL clause of the relevant Report Description entry.

DETAIL (DE) is a report group that is printed each time a GENERATE statement that refers to a detail report group is executed in the Procedure Division. Each detail report group must have a unique data-name at the 01 level in a report (see line 01740 of Figure RW-1). (Note that other report groups will be printed automatically, when applicable, when the GENERATE statement is executed.)

CONTROL FOOTING (CF) has two options:

CF data-name-2 is a report group that is printed following the control break designated by data-name-2 or by a higher level control (see line 01820 of Figure RW-1). There may be only one report group of this type for each control data-name.

CF FINAL is a report group that is printed only once, at the end of the report before the report footing (see line 01960 of Figure RW-1). There may be only one report group of this type in a report.

If this type of report group is used, data-name-2 and/or FINAL must be cited in the CONTROL clause of the relevant Report Description entry.

OVERFLOW FOOTING (OV) is a report group that is printed at the end of a page, following an overflow condition. There may be only one report group of this type in a report (see line 02000 of Figure RW-1). If OV is specified, the PAGE clause, including LAST DETAIL integer-e must be given in the Report Description entry for this report.

PAGE FOOTING (PF) is a report group that is printed at the end of a page, following a page condition. There may be only one report group of this type in a report (see line 02060 If Figure RW-2. of PAGE FOOTING is written, the PAGE clause must appear in the Report Description entry containing this Report Group Description entry.

REPORT FOOTING (RF) is a report group that is printed only once, at the end of the report. There may be only one report group of this type in a report (see line 02090 of Figure RW-1).

Page and Overflow Conditions

If the current report group exceeds the limitation of LAST DETAIL integer-e (described earlier in the subsection "PAGE Clause"), the following object-time rules apply:

1. If the current report group is a detail report group and controls are specified, an overflow condition exists.
2. If the current report group is a control heading, a page condition exists.
3. If the current report group is a control footing, a test is made to determine if the footing can be printed within the limits specified in the PAGE clause. If the footing can be printed without exceeding the established limits, the footing is printed and a page condition exists. If the footing cannot be printed within the established limits, an overflow condition exists.
4. Page footing and page heading, if specified, are printed whenever a page condition exists.
5. Overflow footing and overflow heading, if specified, are printed whenever an overflow condition exists. If not specified, the page footing and page heading are printed. If page footing and page heading are not specified, no footing or heading is printed.

REPORT ELEMENT DESCRIPTION ENTRY

A Report Element Description entry is used to describe a group item or an elementary item within a report group. Report Element Description entries must be written in the order in which the items they describe are to be printed (from left to right). An item whose description contains subordinate entries is called a group item. An item whose description does not contain subordinate entries is called an elementary item. The format of the Report Element Description entry is:

```
level-number [data-name] [LINE-clause] [COLUMN-clause]

     [GROUP INDICATE] [BLANK-clause]

     [RESET-clause] [PICTURE-clause]

     {SOURCE-clause }
     {SUM-clause    } [JUSTIFIED RIGHT]
     {VALUE-clause  }
```

Level-number can be from 02 to 49 and is required.

Level-numbers 02 to 49 identify Report Element Description entries. Subordinate items must have higher (but not necessarily successive) level-numbers that are the items to which they are subordinate. Figure RW-1 5 shows a Report Element Description entry with an 02 level-number.

Data-name is the name of the report item being described. Data-name, if specified, must follow the level number. Data-name is used only if there are references to the elementary item (e.g., SUM clause) in the Procedure Division or elsewhere in the Data Division. Data-names are qualified automatically by higher-level data-names and report-names. Therefore, data-names need not be unique within report group descriptions if a unique higher-level name can be used for qualification.

The BLANK clause is described in Section 5 in the subsection "BLANK Clause."

PICTURE is a required clause and is described in Section 5 in the subsection "PICTURE Clause."

VALUE is described in Section 5 in the subsection "VALUE Clause."

JUSTIFIED RIGHT is described in Section 5 in the subsection "JUSTIFIED RIGHT." RESET, SOURCE,SUM, LINE, COLUMN, and GROUP INDICATE are described in the following text.

The level-number and either the SOURCE, SUM, or VALUE clause are required. The PICTURE clause is required for elementary items, except when the COLUMN clause does not appear. The LINE clause is required only if it does not appear in the higher-level Report Group Description entry or in a previous Report Element Description entry for the same report group at an equal or higher level. All other clauses shown in the Report Element Description entry format are optional.

The LINE clause is described in the subsection "LINE Clause" under "Report Group Description Entry." Note that a LINE clause must be specified at or before the entry for the first elementary item of a report group.

COLUMN Clause

The COLUMN clause is used to indicate the placement of an elementary item within a print line. If this clause is not specified, the item is not printed.The COLUMN clause may only appear in Report Element Description entries that define elementary items. The format of the COLUMN clause is:

```
[COLUMN integer]
```

Integer specifies the position in the print line where the leftmost (high-order) character of the elementary item is to be placed.

The following example shows how this clause can be coded:

```
02 COLUMN 41 PICTURE  A(9) VALUE IS 'CONTINUED'.
```

Line 01690 of Figure RW-1 shows how this coding is incorporated in a typical source program. In this example, CONTINUED will be printed columns 41 through 49.

GROUP INDICATE

The GROUP INDICATE clause is used when an item is to be printed only once following a control break or at the beginning of a new page. The format of the GROUP INDICATE clause is:

[GROUP INDICATE]

The following example shows how this clause can be coded:

```
02 COLUMN 13  GROUP INDICATE PICTURE 99 SOURCE DAY.
```

In Figure RW-2 the day of the month is printed only on the first detail line of each group, because the GROUP INDICATE clause was used in line 01770 of Figure RW-1.

RESET Clause

The RESET clause is used when the programmer wants to override the automatic resetting of the sum counters after an associated control break. If RESET is not specified, the sum counters are reset to zero automatically, when an associated control break occurs. It can only appear at elementary level in association with a SUM clause.

The format of the RESET clause is:

```
[RESET ON {data-name}]
          {FINAL    }
```

Data-name is the name of a higher-level control and must be one of the data-names listed in the CONTROL clause of the Report Description entry. This clause can be used if the programmer wants cumulative totals to appear at the control break associated with data-name.

FINAL is used to indicate that the counter is not to be reset until the final control footing has been printed. This clause can be used if the programmer wants cumulative totals to appear throughout the report.

The following example shows how this clause can be coded:

```
02 COLUMN 65 PICTURE $$$$9.99 SUM COST
   RESET ON FINAL.
```

Line 01890 of figure RW-1 shows how this coding is incorporated in a typical source program.

SOURCE Clause and SUM Clause

The Report Writer "gathers" the required data and prepares it for output according to its description in a PICTURE clause. The SOURCE clause is used to locate the data to be gathered. The Report Writer will also accumulate data to produce specified totals. The SUM clause is used to specify the data to be totaled and may only appear in a control footing report group. These clauses can only be given at the elementary level.

The format of the SOURCE clause is:

SOURCE data-name

Data-name is the name of the data to be reported and represents the current values of the data. Data-name must be described in the file, Working-Storage, or Linkage Section of the Data Division. Data-name may be subscripted, if applicable, and must be qualified where necessary to make it unique.

The following example shows how this clause can be coded:

```
02 LINE 5 COLUMN 32 PICTURE  A(9)
   SOURCE MONTHNAME OF RECORD-AREA (MONTH).
```

Line 01670 of Figure RW-1 shows how this coding is incorporated in a typical source program.

The format of the SUM clause is:

SUM data-name-1... [UPON data-name-n]

The data-names following SUM are the names of the items that are to be summed. Each item being summed must appear as the operand of a SOURCE clause in a detail report group or must be the name of an item containing a SUM clause in an equal or lower-level control report group.

The UPON option is used for selective summation of the data items. Data-name-n must be the name of a detail report group. When this option is used, the values of data-name-1, etc. are added to the sum counter only when the detail report group designated by the data-name has been referred to by a GENERATE statement in the Procedure Division. When the UPON option is not used, each detail group reference causes all specified summations. This causes the sum counter for the major total to be updated after the minor control break and before the major sum counter is reset.

If major and minor sums are to be labeled the name of a sum counter at a lower control level may be the operand of a SUM clause at a higher control level. The operands of the SUM clause may be subscripted, if applicable, and must be qualified where necessary to make the named items unique.

The following example shows how this clause can be coded:

```
02 COLUMN 30 PICTURE ZZ9 SUM NO-PURCHASES.
```

Line 01870 of Figure RW-1 shows how this coding is incorporated in a typical source program.

PAGE AND LINE COUNTERS

PAGE-COUNTER is a fixed data-name specifying a compiler-generated counter to be used by the Report Writer. Each time a new page is started, PAGE-COUNTER is automatically incremented by one. PAGE-COUNTER may be referred to by Procedure Division statements. A page counter is generated for each report-name when more than one report entry appears in the Report Section. Therefore, it must be qualified by the report-name when more than one report description entry appears in the Report Section.

PAGE-COUNTER is set to one initially. If a starting value other than one is desired, it may be set by a Procedure Division statement after the INITIATE statement has been executed. PAGE-COUNTER is incremented automatically by one each time a page break is recognized, after producing page or overflow footings (if applicable), but before producing any page or overflow headings.

LINE-COUNTER is a fixed data-name. It is generated by the Report Writer and used to determine when page headings and footings should be produced. LINE-COUNTER may be referred to by Procedure Division statements. Since a line counter is generated for each report described in the Report Section, LINE-COUNTER must be qualified by the report-name when more than one report entry appears in the Report Section.

LINE-COUNTER is set to zero initially and is reset to zero after every page break. No additional setting takes place based on relative line numbering after the counter is set to zero. At any given time, the value of LINE-COUNTER represents one of the following.

1. The last line number produced from the previously generated report group.

2. The last line number skipped to by a previous NEXT GROUP specification.

LINE-COUNTER is used by the Report Writer for test and control purposes. Therefore, programmers should be cautious about changing the value of LINE-COUNTER by Procedure Division statements, as the ensuing page format control may be unpredictable.

PROCEDURE DIVISION CONSIDERATIONS

To produce a report, the INITIATE, GENERATE, AND TERMINATE statements must be used in the Procedure Division. In addition, a USE BEFORE REPORTING declarative may be written in the Declaratives portion of the Procedure Division. This option allows the programmer to manipulate or alter data immediately before it is printed.

INITIATE Statement

The INITIATE statement is used to initialize all counters preparatory to producing a report. The format of the INITIATE statement is as follows:

```
          {ALL          }
INITIATE  {report-name ...}
```

ALL specifies that all report-names defined in the Report Description entries of the Report Section are to be initiated.

The report-names are the names of the reports to be initiated. These reports must be defined by Report Description entries in the Report Section of the Data Division.

Only one INITIATE statement can be executed for each report-name. The INITIATE statement does not open the file with which the report is associated. As shown in the coding example, an OPEN statement for that file must be given.

```
OPEN INPUT INFILE, OUTPUT REPORT-FILE.
INITIATE EXPENSE-REPORT.
```

Lines 02140 and 02150 of Figure RW-1 show how this coding is incorporated in a typical source program.

GENERATE Statement

The GENERATE statement is used to produce a report. The format of the GENERATE statement is:

GENERATE data-name

Data-name can be the name of a report or the name of a detail report group.

The GENERATE statement produces the following operations:

1. Produces the detail line, if data-name is the name of a detail report group. If data-name is the name of a report, no detail line is produced.

2. Produces the appropriate page or overflow footings and/or headings.

3. Recognizes specified control breaks and produces appropriate control footings and/or headings.

4. Updates each set of sum counters for each detail line and resets the counters on an associated control break.

When the first GENERATE statement referring to a report or a detail report group is executed, specified report headings and page headings are produced. Then, all specified control headings are produced in the order FINAL, major, intermediate, minor, and are followed immediately by any detail report group specified in the statement. If a control break is recognized when a GENERATE statement is executed(other than the first execution), the specified control footings are produced, from the minor report group up to and including the report group specified for the data-name that caused the control break. Then, the specified control headings are produced, from the report group specified for the data-name that caused the control break down to the minor report group. The detail report group specified in the GENERATE statement is then produced.

Under control of the Report Writer, data is moved into the data area described in the Report Group Description entry according to the rules described in Section 6 under the subsection "MOVE."

The following example shows how this statement can be coded:

```
READATA.  READ INFILE AT END GO TO COMPLETE.
    GENERATE DETAIL-LINE.  GO TO READATA.
```

Lines 02160 and 02170 of Figure RW-1 show how this coding is incorporated in a typical source program.

TERMINATE Statement

The TERMINATE statement is used to end a report. It produces all of the control footings associated with the report and completes the Report Writer functions. The control footings are the same as if a control break occurred at the highest level. The format of the TERMINATE statement is:

```
TERMINATE {ALL          }
          {report-name...}
```

ALL is used to terminate all previously initiated reports. The report-names are used when the programmer wishes to terminate only certain previously initiated reports. Each report-name given in a TERMINATE statement must be defined by a Report Description entry in the Data Division.

The TERMINATE statement produces the appropriate page footings and report footings, but does not close the file with which the report is associated. As shown in the coding example, a CLOSE statement for the file must be executed subsequent to execution of a TERMINATE statement. Only one TERMINATE statement can be executed for each report-name.

The following example shows how this statement can be coded:

```
COMPLETE.  TERMINATE EXPENSE-REPORT.
    CLOSE INFILE, REPORT-FILE.  STOP RUN.
```

Lines 02180 and 02190 of Figure RW-1 show how this coding is incorporated in a typical source program.

USE BEFORE REPORTING Declarative

The USE BEFORE REPORTING declarative is used to perform procedures immediately before the specified report group is produced. It can also be used to suppress printing of the specified report group. The format of the USE BEFORE REPORTING declarative is:

section-name SECTION. USE BEFORE REPORTING data-name.

Data-name is the name of a report group (other than a detail report group).

When the user wishes to suppress the printing of the specified report group, the statement

```
Ext MOVE 1 to PRINT-SWITCH
```

is used in the USE BEFORE REPORTING declarative. When this statement is encountered, only the specified report group is not printed; the statement must be written for each report group whose printing is to be suppressed.

The following coding example illustrates the use and operation of this declarative. Part 1 of the example shows the definition of two control footing report groups in the Report Section of the Data Division. Part 2 of the example shows a method of using the USE BEFORE REPORTING declarative. In this example, the USE BEFORE REPORTING declarative in Part 2 will be executed before the MINOR control footing (described in Part 1) is produced.

Part 1

```
01  MINOR TYPE CONTROL FOOTING C-1 LINE PLUS 1.
    32 A SUM P PICTURE 9(9)
    02 B SUM Q PICTURE 9(9).
    02 C SOURCE E PICTURE 9(9).
01 MAJOR TYPE CONTROL FOOTING C-2 LINE PLUS 2.
    02 R SUM A PICTURE 9(9).
```

P and Q are defined in the File Section and E is defined in the Working-Storage Section.

Part 2

```
DECLARATIVES.
RW SECTION.  USE BEFORE REPORTING MINOR.
    COMPUTE E =P / Q.
    MOVE E TO C OF MINOR.
    IF E LESS 5, MOVE ZERO TO A.
END DECLARATIVES.
```

When a control break occurs for C-1, the following operations are performed:

1. The value in P is added to the value in A, the value in Q is added to the value in B, and the value in E is moved to C.

2. The USE BEFORE REPORTING declarative is executed.

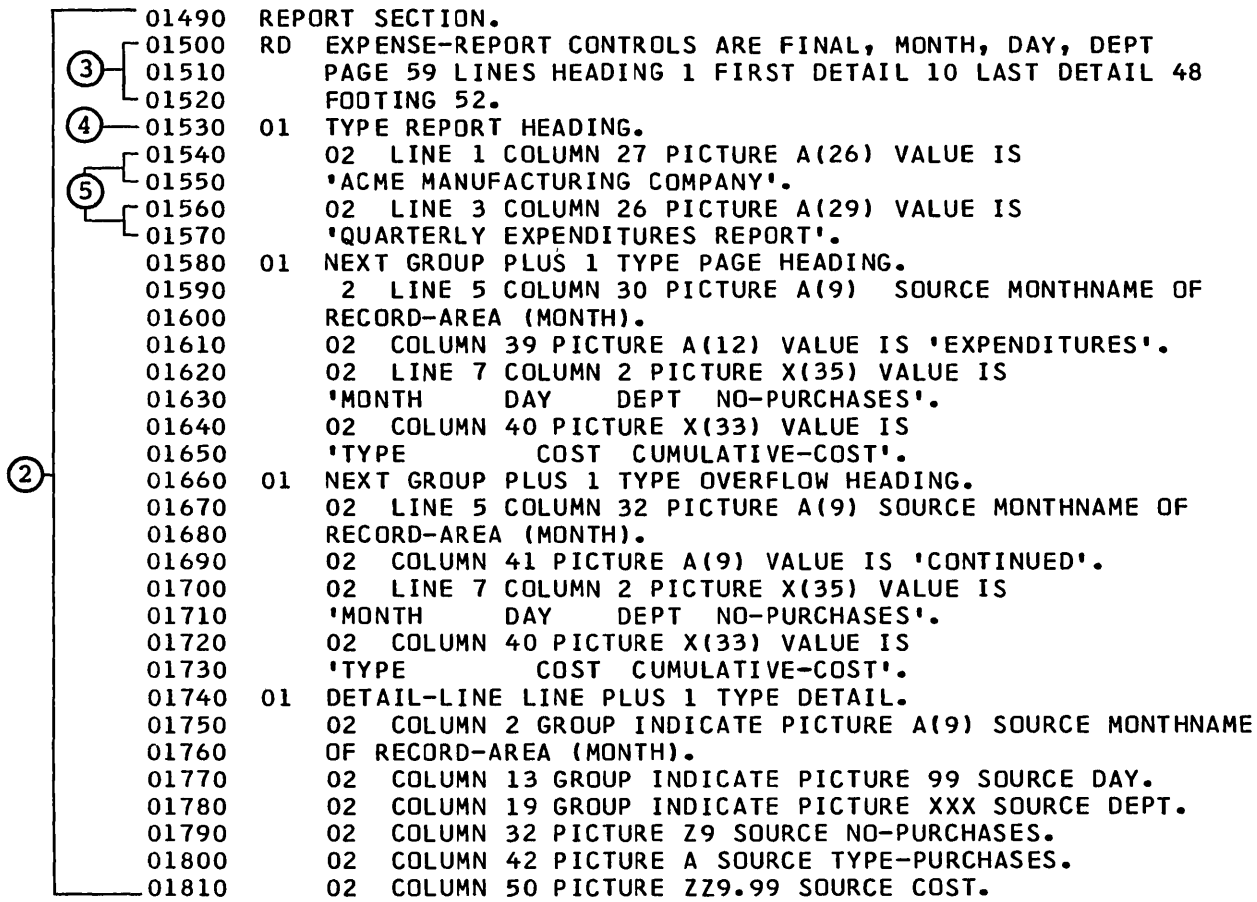
3. The values in A,B, and C are each edited (if specified), moved to the print area, and printed.

4. The value in A is added to the value in R.

5. A and B are reset to zero.

```
01010  IDENTIFICATION DIVISION.
01020  PROGRAM-ID. ACME.
01030  INSTALLATION. ACME ACCOUNTING DEPARTMENT.
01040  REMARKS.  THE REPORT WAS PRODUCED BY THE REPORT WRITER FEATURE.
01050
01060  ENVIRONMENT DIVISION.
01070  CONFIGURATION SECTION.
01080  SOURCE-COMPUTER.  IBM-360 F50.
01090  OBJECT-COMPUTER.  IBM-360 F50.
01100  INPUT-OUTPUT SECTION.
01110  FILE-CONTROL.
01120      SELECT INFILE ASSIGN TO UTILITY. SELECT REPORT-FILE ASSIGN TO
01130      UTILITY.
01140
01150  DATA DIVISION.
01160  FILE SECTION.
01170  FD  INFILE BLOCK CONTAINS 25 RECORDS VALUE OF FILE-ID 'ACME0016'.
01180  01  INPUT-RECORD.
01190      02  FILLER          PICTURE AA.
01200      02  DEPT            PICTURE XXX.
01210      02  FILLER          PICTURE AA.
01220      02  NO-PURCHASES    PICTURE 99.
01230      02  FILLER          PICTURE A.
01240      02  TYPE-PURCHASE   PICTURE A.
01250      02  MONTH           PICTURE 99.
01260      02  DAY             PICTURE 99.
01270      02  FILLER          PICTURE A.
01280      02  COST            PICTURE 999V99.
(1)-01290  FD  REPORT-FILE VALUE OF FILE-ID 'ACME0133'  REPORT IS
   01300      EXPENSE-REPORT.
01310  WORKING-STORAGE SECTION.
01320  01  FILLER.
01330      02  RECORD-MONTH.
01340          03  FILLER PICTURE A(9) VALUE IS 'JANUARY  '.
01350          03  FILLER PICTURE A(9) VALUE IS 'FEBRUARY '.
01360          03  FILLER PICTURE A(9) VALUE IS 'MARCH    '.
01370          03  FILLER PICTURE A(9) VALUE IS 'APRIL    '.
01380          03  FILLER PICTURE A(9) VALUE IS 'MAY      '.
01390          03  FILLER PICTURE A(9) VALUE IS 'JUNE     '.
01400          03  FILLER PICTURE A(9) VALUE IS 'JULY     '.
01410          03  FILLER PICTURE A(9) VALUE IS 'AUGUST   '.
01420          03  FILLER PICTURE A(9) VALUE IS 'SEPTEMBER'.
01430          03  FILLER PICTURE A(9) VALUE IS 'OCTOBER  '.
01440          03  FILLER PICTURE A(9) VALUE IS 'NOVEMBER '.
01450          03  FILLER PICTURE A(9) VALUE IS 'DECEMBER '.
01470      02  RECORD-AREA REDEFINES RECORD-MONTH OCCURS 12 TIMES.
01480          03  MONTHNAME PICTURE A(9).
```

FIGURE RW-1. COBOL PROGRAM WITH REPORT WRITER FEATURE. (Continued)



```
01490  REPORT SECTION.
01500  RD  EXPENSE-REPORT CONTROLS ARE FINAL, MONTH, DAY, DEPT
01510      PAGE 59 LINES HEADING 1 FIRST DETAIL 10 LAST DETAIL 48
01520      FOOTING 52.
01530  01  TYPE REPORT HEADING.
01540      02  LINE 1 COLUMN 27 PICTURE A(26) VALUE IS
01550      'ACME MANUFACTURING COMPANY'.
01560      02  LINE 3 COLUMN 26 PICTURE A(29) VALUE IS
01570      'QUARTERLY EXPENDITURES REPORT'.
01580  01  NEXT GROUP PLUS 1 TYPE PAGE HEADING.
01590       2  LINE 5 COLUMN 30 PICTURE A(9)  SOURCE MONTHNAME OF
01600      RECORD-AREA (MONTH).
01610      02  COLUMN 39 PICTURE A(12) VALUE IS 'EXPENDITURES'.
01620      02  LINE 7 COLUMN 2 PICTURE X(35) VALUE IS
01630      'MONTH     DAY    DEPT  NO-PURCHASES'.
01640      02  COLUMN 40 PICTURE X(33) VALUE IS
01650      'TYPE        COST  CUMULATIVE-COST'.
01660  01  NEXT GROUP PLUS 1 TYPE OVERFLOW HEADING.
01670      02  LINE 5 COLUMN 32 PICTURE A(9) SOURCE MONTHNAME OF
01680      RECORD-AREA (MONTH).
01690      02  COLUMN 41 PICTURE A(9) VALUE IS 'CONTINUED'.
01700      02  LINE 7 COLUMN 2 PICTURE X(35) VALUE IS
01710      'MONTH     DAY    DEPT  NO-PURCHASES'.
01720      02  COLUMN 40 PICTURE X(33) VALUE IS
01730      'TYPE        COST  CUMULATIVE-COST'.
01740  01  DETAIL-LINE LINE PLUS 1 TYPE DETAIL.
01750      02  COLUMN 2 GROUP INDICATE PICTURE A(9) SOURCE MONTHNAME
01760      OF RECORD-AREA (MONTH).
01770      02  COLUMN 13 GROUP INDICATE PICTURE 99 SOURCE DAY.
01780      02  COLUMN 19 GROUP INDICATE PICTURE XXX SOURCE DEPT.
01790      02  COLUMN 32 PICTURE Z9 SOURCE NO-PURCHASES.
01800      02  COLUMN 42 PICTURE A SOURCE TYPE-PURCHASES.
01810      02  COLUMN 50 PICTURE ZZ9.99 SOURCE COST.
```


FIGURE RW-1. COBOL PROGRAM WITH REPORT WRITER FEATURE. (Continued)

```
01820  01  LINE PLUS 2 NEXT GROUP PLUS 1 TYPE CONTROL FOOTING DAY.
01830      02  COLUMN 2 PICTURE X(22) VALUE IS 'PURCHASES AND COST FOR'.
01840      02  COLUMN 24 PICTURE Z9  SOURCE MONTH.
01850      02  COLUMN 26 PICTURE X VALUE IS '-'.
01860      02  COLUMN 27 PICTURE 99 SOURCE DAY.
01870      02  COLUMN 30 PICTURE ZZ9 SUM NO-PURCHASES.
01880      02  MIN COLUMN 49 PICTURE $$$9.99 SUM COST.
01890      02  COLUMN 65 PICTURE $$$$9.99 SUM COST RESET ON FINAL.
(11)-01900 01  LINE PLUS 1 NEXT GROUP PLUS 1 TYPE CONTROL FOOTING MONTH.
01910      02  COLUMN 16 PICTURE A(14) VALUE IS 'TOTAL COST FOR'.
01920      02  COLUMN 31 PICTURE A(9) SOURCE MONTHNAME OF RECORD-AREA
01930      (MONTH).
01940      02  COLUMN 39 PICTURE AAA VALUE IS 'WAS'.
01950      02  INT COLUMN 46 PICTURE $$$9.99 SUM MIN.
01960  01  NEXT GROUP PLUS 1 TYPE CONTROL FOOTING FINAL.
01970      02  LINE PLUS 1 COLUMN 16 PICTURE A(26) VALUE IS
(2)-01980  'TOTAL COST FOR QUARTER WAS'.
01990      02  COLUMN 45 PICTURE $$$$9.99 SUM INT.
02000  01  TYPE OVERFLOW FOOTING.
02010      02  LINE 55 COLUMN 32 PICTURE A(9) SOURCE MONTHNAME OF
02020      RECORD-AREA (MONTH).
02030      02  COLUMN 41 PICTURE A(9) VALUE IS 'CONTINUED'.
02040      02  LINE 57 COLUMN 66 PICTURE X(5) VALUE IS 'PAGE-'.
02050      02  COLUMN 70 PICTURE 99 SOURCE PAGE-COUNTER.
02060  01  TYPE PAGE FOOTING.
(10)-02070 02  LINE 57 COLUMN 66 PICTURE X(5) VALUE IS 'PAGE-'.
02080      02  COLUMN 70 PICTURE 99 SOURCE PAGE-COUNTER.
02090  01  TYPE REPORT FOOTING.
02100      02  LINE PLUS 1 COLUMN 32 PICTURE A(13) VALUE IS
02110      'END OF REPORT'.
02120
02130  PROCEDURE DIVISION.
02140      OPEN INPUT INFILE, OUTPUT REPORT-FILE.
(6)-02150  INITIATE EXPENSE-REPORT.
02160  READATA. READ INFILE AT END GO TO COMPLETE.
```

FIGURE RW-1. COBOL PROGRAM WITH REPORT WRITER FEATURE.(Continued)

```
(7)-02170       GENERATE DETAIL-LINE. GO TO READATA.
(8)-02180   COMPLETE. TERMINATE EXPENSE-REPORT.
   02190       CLOSE INFILE, REPORT-FILE. STOP RUN.
```

```
(9)-A00   2 A0101   200
   A00   4 B0101   624
   A00   1 C0113   800
   A00  10 D0115  1920
   A00   5 A0126   500
   A00  15 C0127 12000
   A00   6 B0127   936
   A00   2 B0130   312
   A00   1 A0131   100
   A01   2 B0102   200
   A01   2 B0105   200
   A01   2 B0106   200

         .
         .
         .

   A07  10 F0216  2500
   A07  10 E0217  3000
   A07  10 G0217  5000
```

KEY

```
 1  FILE DESCRIPTION ENTRY
 2  REPORT SECTION OF DATA DIVISION
 3  REPORT DESCRIPTION ENTRY
 4  REPORT GROUP DESCRIPTION ENTRY
 5  REPORT ELEMENT DESCRIPTION ENTRY
 6  INITIATE STATEMENT
 7  GENERATE STATEMENT
 8  TERMINATE STATEMENT
 9  INPUT DATA
10  ELEMENTARY ITEMS
11  CONTROL FOOTING ENTRY
```

FIGURE RW-1. COBOL PROGRAM WITH REPORT WRITER FEATURE. (Continued)

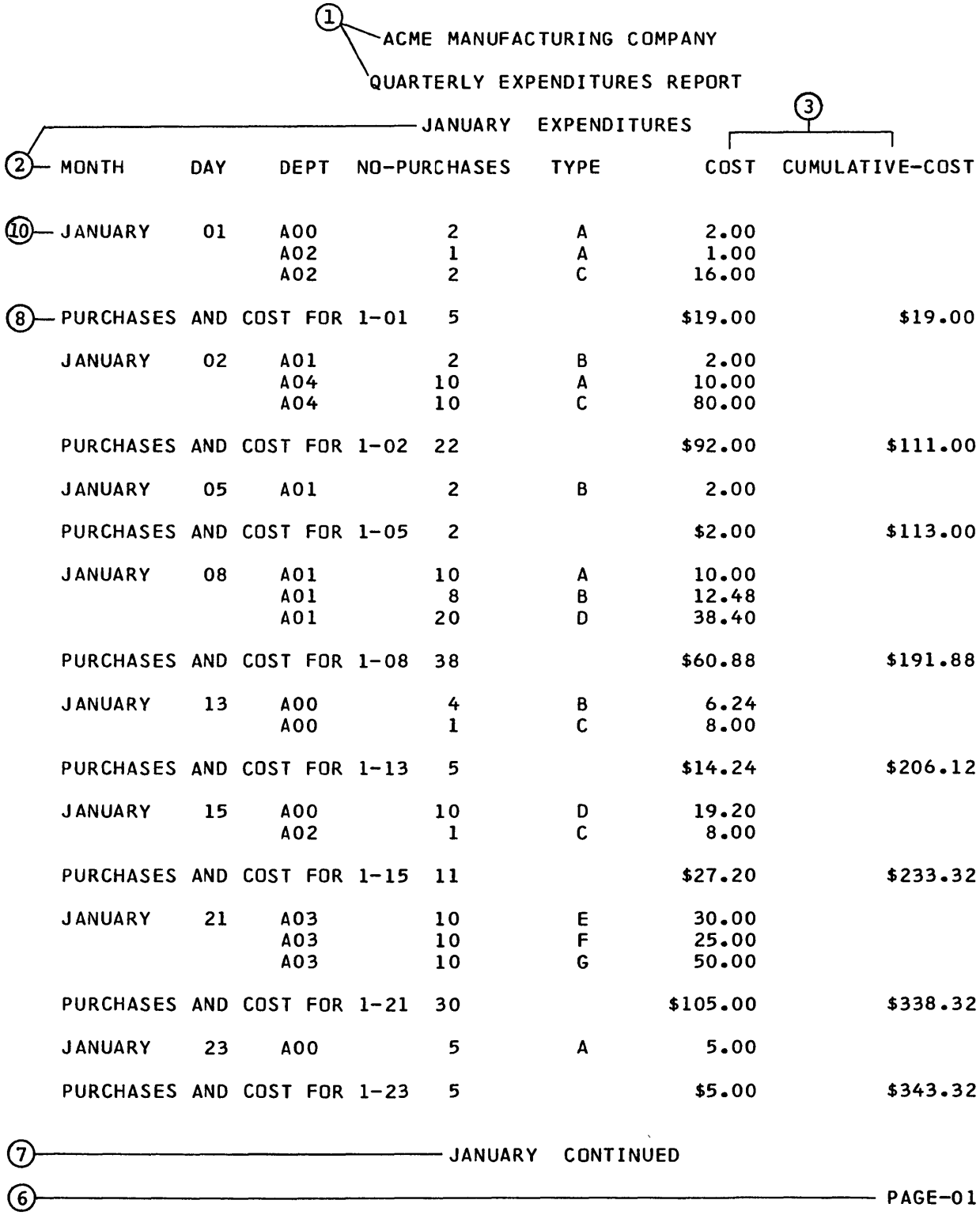


FIGURE RW-2. REPORT PRODUCED BY REPORT WRITER FEATURE. (Continued)

④ JANUARY CONTINUED

MONTH	DAY	DEPT	NO-PURCHASES	TYPE	COST	CUMULATIVE-COST
JANUARY	26	A04	5	A	5.00	
		A04	5	B	7.80	
PURCHASES AND COST FOR 1-26			10		$12.80	$356.12
JANUARY	27	A00	6	B	9.36	
		A00	15	C	120.00	
PURCHASES AND COST FOR 1-27			21		$129.36	$485.48
JANUARY	30	A00	2	B	3.12	
		A02	10	A	10.00	
		A02	1	C	8.00	
		A04	15	B	23.40	
		A04	10	C	80.00	
PURCHASES AND COST FOR 1-30			38		$124.52	$610.00
JANUARY	31	A00	1	A	1.00	
		A04	6	A	6.00	
PURCHASES AND COST FOR 1-31			7		$7.00	$617.00

⑨ TOTAL COST FOR JANUARY WAS $617.00

FIGURE RW-2. REPORT PRODUCED BY REPORT WRITER FEATURE. (Continued)

(2) FEBRUARY EXPENDITURES

MONTH	DAY	DEPT	NO-PURCHASES	TYPE	COST	CUMULATIVE-COST
FEBRUARY	15	A02	10	A	10.00	
		A02	2	B	3.12	
		A02	1	C	8.00	
		A03	15	G	75.00	
		A04	5	B	7.80	
		A05	8	A	8.00	
		A05	5	C	40.00	
PURCHASES AND COST FOR 2-15			46		$151.92	$768.92
FEBRUARY	16	A02	2	C	16.00	
		A06	10	A	10.00	
		A07	10	A	10.00	
		A07	10	F	25.00	
PURCHASES AND COST FOR 2-16			32		$61.00	$829.92
FEBRUARY	17	A07	10	E	30.00	
		A07	10	G	50.00	
PURCHASES AND COST FOR 2-17			20		$80.00	$909.92
FEBRUARY	21	A06	20	A	20.00	
		A06	20	B	31.20	
		A06	20	C	160.00	
		A06	20	D	38.40	
		A06	20	E	60.00	
		A06	20	F	50.00	
		A06	20	G	100.00	
PURCHASES AND COST FOR 2-21			140		$459.60	$1369.52
FEBRUARY	27	A01	21	D	40.32	
PURCHASES AND COST FOR 2-27			21		$40.32	$1409.84
FEBRUARY	28	A02	3	B	4.68	
		A02	5	C	40.00	
		A03	15	E	45.00	
PURCHASES AND COST FOR 2-28			23		$89.68	$1499.52

TOTAL COST FOR FEBRUARY WAS $882.52

FIGURE RW-2. REPORT PRODUCED BY REPORT WRITER FEATURE. (Continued)

MARCH EXPENDITURES

MONTH	DAY	DEPT	NO-PURCHASES	TYPE	COST	CUMULATIVE-COST
MARCH	01	A02	5	A	5.00	
			1	C	8.00	
		A03	25	G	125.00	
PURCHASES AND COST FOR 3-01			31		$138.00	$1637.52
MARCH	06	A02	5	A	5.00	
PURCHASES AND COST FOR 3-06			5		$5.00	$1642.52
MARCH	07	A02	5	A	5.00	
PURCHASES AND COST FOR 3-07			5		$5.00	$1647.52
MARCH	13	A02	10	A	10.00	
PURCHASES AND COST FOR 3-13			10		$10.00	$1657.52
MARCH	15	A01	21	A	21.00	
		A02	1	A	1.00	
		A03	15	F	37.50	
		A06	5	E	15.00	
			5	F	12.50	
PURCHASES AND COST FOR 3-15			47		$87.00	$1744.52
MARCH	20	A03	15	E	45.00	
PURCHASES AND COST FOR 3-20			15		$45.00	$1789.52
MARCH	21	A02	15	A	15.00	
		A03	15	F	37.50	
PURCHASES AND COST FOR 3-21			30		$52.50	$1842.02
MARCH	23	A02	2	A	2.00	
PURCHASES AND COST FOR 3-23			2		$2.00	$1844.02

FIGURE RW-2. REPORT PRODUCED BY REPORT WRITER FEATURE. (Continued)

MARCH CONTINUED

MONTH	DAY	DEPT	NO-PURCHASES	TYPE	COST	CUMULATIVE-COST
MARCH	25	A03	30	F	75.00	
PURCHASES AND COST FOR 3-25			30		$75.00	$1919.02
MARCH	26	A02	1	A	1.00	
PURCHASES AND COST FOR 3-26			1		$1.00	$1920.02
MARCH	29	A01	6	C	48.00	
PURCHASES AND COST FOR 3-29			6		$48.00	$1968.02
MARCH	31	A03	20	E	60.00	
			10	G	50.00	
PURCHASES AND COST FOR 3-31			30		$110.00	$2078.02

TOTAL COST FOR MARCH WAS $578.58

(5) TOTAL COST FOR QUARTER WAS $2078.02

(11) END OF REPORT

FIGURE RW-2. REPORT PRODUCED BY REPORT WRITER FEATURE. (Continued)

KEY

1 REPORT HEADING
2 PAGE HEADING
3 GROUP INDICATE ITEMS
4 OVERFLOW HEADING
5 REPORT FOOTING
6 PAGE FOOTING
7 OVERFLOW FOOTING
8 INTERMEDIATE CONTROL FOOTING
9 MAJOR CONTROL FOOTING
10 DETAIL LINE
11 FINAL CONTROL FOOTING

FIGURE RW-2. REPORT PRODUCED BY REPORT WRITER FEATURE. (Continued)

F Only SECTION 8: SORT FEATURE

The Sort feature permits the programmer to include sort operations in a COBOL program. The Sort feature also includes the optional use of input and output procedures that can alter the data during the sorting operation.

ILLUSTRATIONS

Figure S-1 shows the flow of data through the sorting operation. Figure S-2 is an example of a typical sorting problem written in System/360 COBOL. The items discussed in this section have been keyed within the figure.

BASIC SORT CONCEPTS

A sort program is used to rearrange records according to one or more specific fields in each record. These fields are called sort-keys. For example, if a file contains individual records for each employee in a firm, the programmer may want to rearrange these records in alphabetic order according to the employees' names. The field in each record containing the employee's name would then be a sort-key.

In a sort operation, records are read from the input-file and placed in blocks of contiguous records (called strings) on the sort-file. Sort-file is the generic term for several intermediate working files. The programmer need not be concerned with these intermediate files, since they are used by the internal Sort/Merge Program to perform the actual sorting and merging operation. To the COBOL programmer, the sort operation is automatic, and he can consider the sort-file to be one file on which sorting and merging are accomplished.

When all of the records on the input-file have been transferred to the sort-file, the records in each string are merged, forming longer strings. The final operation merges the strings and places the sorted records on the output-file.

ELEMENTS OF THE SORT FEATURE

There are three basic elements of the Sort feature: the SORT statement, the input procedure, and the output procedure.

The SORT statement is used in the main body of the Procedure Division. It specifies the file to be sorted, the sort-keys in order of importance, whether the sequence controlled by each sort-key is ascending or descending, and whether input and/or output procedures are to intervene in the sorting operation. A detailed description of the SORT statement format is in the subsection "SORT Statement" under "Procedure Division Considerations."

The input and output procedures are optional. The input and output procedures must be separate sections and may not refer to each other or to other sections or paragraphs. They may only be referred to by a SORT statement. Control is passed to each procedure only by execution of the SORT statement.

Both the input and output procedures can be used to manipulate data. The input procedure can be used to select, create, and merge certain records from one or more files, for sorting; the output procedure can be used to process sorted records and to place them on one or more output files.

If INPUT PROCEDURE is specified in the SORT statement, control is passed to the input procedure. The input procedure can contain any statements to read, select, create, or modify records to be released to the sort-file. If INPUT PROCEDURE is not specified in the SORT statement, all of the records in the input-file are transferred to the sort-file when the SORT statement is executed. If INPUT PROCEDURE is specified in the SORT statement, the input procedure is performed as soon as the SORT statement is executed, before the records are placed on the sort-file. The input procedure must contain at least one RELEASE statement, which puts the selected records in the sort-file. The sorting operation begins when the input procedure terminates execution. If, however, no input procedure has been specified, it begins at once.

If OUTPUT PROCEDURE is specified in the SORT statement, the output procedure is performed in conjuction with the final merge If OUTPUT PROCEDURE is not specified in the SORT statement, the final merge is accomplished and the sorted records are placed on the specified output-file. The output procedure can contain any statements to select, modify, copy, or write the sorted records that are returned, one at a time, from the sort-file. Before a sorted record can be processed, it must be retrieved from the sort-file by executing a RETURN statement in the output procedure. The output procedure must contain at least one RETURN statement. When the output procedure terminates execution, the sorting operation is completed and control returns to the statement following the SORT statement.

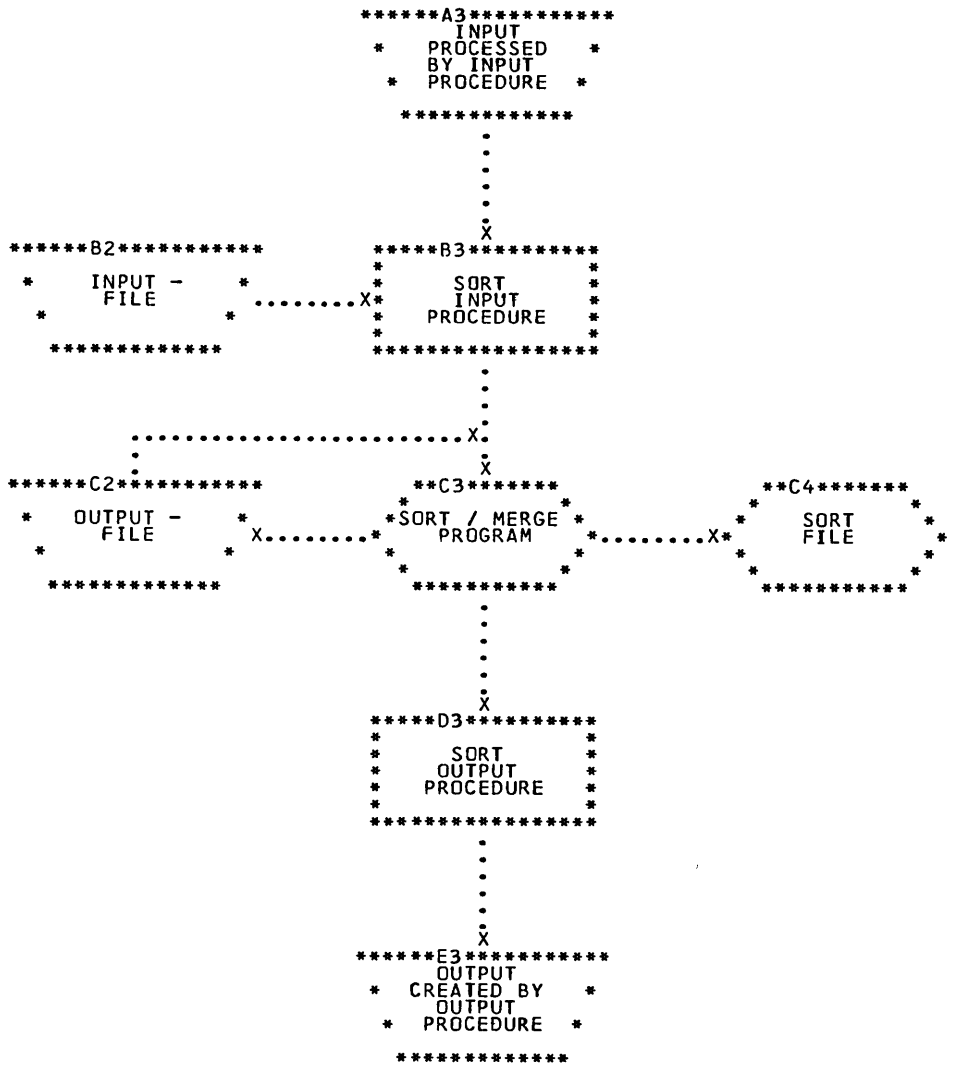


Figure S-1. Flow of Data through a Sort Operation.

SORT-KEYS

A sort-key is that section of a record which is used as the basis for sorting. It must have a fixed length and may be one of the following types of items:

Internal floating-point
External decimal
Internal decimal
Binary
Alphanumeric
Alphabetic

These allowable data types are described in Section 5, under the subsection "Elementary Items."

A sort-key can be located anywhere within the record, with one limitation: the sort-keys of all records to be sorted in the same operation must each have a constant displacement from the beginning of all records. A displacement cannot exceed 4095 bytes.

Data-names used as sort-keys must be unique; they define sort-key locations in all records in a file. For example, for a file named FILEA, A in REC1 may be listed as a sort-key in a SORT statement. Then, this sort-key, in all types of records in FILEA, will be defined by the length and displacement of A in REC1, no matter what names identify the area in other records in the file.

Sort-keys cannot be subscripted. As many as 12 sort-keys can be specified for a file. These keys are specified in the SORT statement in the order of major to minor (i.e., the order in which the programmer wants them to be checked by the sort operation). The total length of all sort-keys in a record must not exceed 256 bytes.

DATA DIVISION CONSIDERATIONS

When the Sort feature is used, Sort Description entries and related Record Description entries must be written in the File Section of the Data Division (in addition to the File Description entries for input-files and/or output files). Sort Description entries describe the sort-files that are to be sorted; the Record Description entries are used to define the data in the sort-file. These entries are described in the following text.

Sort Description Entry

The name of a sort-file that is to be sorted must appear in the File Section in a separate Sort Description entry. The format of this entry is:

```
SD sort-file-name

     VALUE OF FILE-ID IS external-name

     [RECORD CONTAINS [integer-1 TO]

     integer-2 CHARACTERS]

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

SD is the level indicator and is required.

Sort-file-name is the name of a sort-file specified in a SORT statement. The sort-file-name must be unique and is required. Every sort-file named in a Sort Description entry must appear in a SELECT clause in the File-Control paragraph of the Environment Division.

The RECORD CONTAINS clause is optional and is described in Section 5. Only the form shown here is allowed.

Figure S-2 shows Sort Description entry.

Record Description Entry

Each Sort Description entry must be followed by one or more Record Description entries. The Record Description entries are used to

describe the characteristics of each item in the data records that is to appear on the sort-file. Record Description entries are described in Section 5.

Figure S-2 (2) shows a sort-file Record Description entry.

PROCEDURE DIVISION CONSIDERATIONS

When the Sort Feature isused, a SORT statement is written in the main body of the Procedure Division. (See Figure S-2 (3).)

If input and output procedures are used, they are written as separate sections in the main body of the Procedure Division. The RELEASE and RETURN statements are used in the input and output procedures (Figure S-2 (4) (5)); these statements and the SORT statement are described in the following text.

SORT Statement

The format of the SORT statement is:

```
SORT sort-file-name {DESCENDING} data-name-1...
                    {ASCENDING }

[{DESCENDING} data-name-2...] {USING  file-name-1          }
 {ASCENDING }                 {INPUT PROCEDURE section-name-1}

{OUTPUT PROCEDURE section-name-2}
{GIVING  file-name-2             }
```

Sort-file-name is the name of the sort-file associated with the input-file that is to be sorted. It is required and must have a Sort Description entry in the File Section of the Data Division.

The DESCENDING or ASCENDING clause is required. When DESCENDING is specified for one or more sort-keys, the sorted sequence is from highest value to lowest value. Similarly, when ASCENDING is specified, the sorted sequence is from lowest to highest value.

Data-name-1 and data-name-2 designate sort-keys. Data-names are listed in the SORT statement according to their significance.

Section-name-1 is the name of the input procedure.

The USING option is required if there is no input procedure. File-name-1 is the name of the input-file that is to be sorted. If the USING option is specified, all the records in file-name-1 are used as input to the Sort/Merge Program. File-name-1 must not be open when the SORT statement is executed. The SORT statement automatically performs the necessary OPEN, READ, and CLOSE functions for file-name-1. File-name-1 must have a File Description entry (not a Sort Description entry) in the File Section of the Data Division.

Section-name-2 is the name of the output procedure.

The GIVING option is required if there is no output procedure. File-name-2 is the name of the output-file. If the GIVING option is specified, all the records in sort-file-name are merged for output onto file-name-2. File-name-2 must not be open when the SORT statement is executed. The SORT statement automatically opens file-name-2 before transferring the records and closes it after the last record in the sort-file is returned. File-name-2 must have a File Description entry (not a Sort Description entry) in the File Section of the Data Division.

The following example shows how this statement can be written:

```
SORT SORT-FILE-1 ASCENDING FIELD-AA
  DESCENDING FIELD-BB, ASCENDING FIELD-CC
  INPUT PROCEDURE RECORD-SELECTION
  OUTPUT PROCEDURE PROCESS-SORTED-RECORDS.
```

In this example, SORT-FILE-1 is the name of the sort-file associated with the input-file that is to be sorted; FIELD-AA, FIELD-BB, and FIELD-CC are sort-keys; RECORD-SELECTION is the name of the input procedure; PROCESS-SORTED-RECORDS is the name of the output procedure. Figure S-2 (3) shows how this statement is used.

RELEASE Statement

The RELEASE statement is used in the input procedure to transfer records to the initial phase of a sort operation. The format of a RELEASE statement is:

RELEASE record-name

Record-name is the name of an 01-level Sort Description entry associated with the particular sort file.

Figure S-2 (4) shows how this statement is used.

RETURN Statement

The RETURN statement is used in the output procedure to obtain sorted records from the sort-file in conjunction with the final merge.The format of a RETURN statement is:

RETURN sort-file-name [AT END imperative statement ...]

Sort-file-name is the name of a sort-file and has a Sort Description entry in the File Section of the Data Division.

The AT END clause is required. The AT END portion of the RETURN statement is executed when all sorted records have been retrieved.

Figure S-2 (5) shows how this statement is used.

```
01010  IDENTIFICATION DIVISION.
01020  PROGRAM-ID. 360SORT.
01030  REMARKS.  THIS PROGRAM WAS WRITTEN TO DEMONSTRATE THE USE OF
01040            THE SORT FEATURE.  THIS PROGRAM PERFORMS THE FOLLOWING
01050            TASKS -
01060            1.  SELECTS, FROM A FILE OF 1000-CHARACTER RECORDS,
01070                THOSE RECORDS HAVING FIELD-A NOT EQUAL TO FIELD-B.
01080            2.  EXTRACTS INFORMATION FROM THE SELECTED RECORDS.
01090            3.  SORTS THE SELECTED RECORDS INTO SEQUENCE, USING
01100                FIELD-AA, FIELD-BB, AND FIELD-CC AS SORT KEYS.
01110            4.  WRITES THOSE SORTED RECORDS HAVING FIELD-FF EQUAL
01120                TO FIELD-EE ON FILE-3 AND WRITES SELECTED DATA OF
01130                THE OTHER RECORDS ON FILE-2.
01140
01150  ENVIRONMENT DIVISION.
01160  CONFIGURATION SECTION.
01170  SOURCE-COMPUTER.  IBM-360 F50.
01180  OBJECT-COMPUTER.  IBM-360 F50.
01190  INPUT-OUTPUT SECTION.
01200  FILE-CONTROL.  SELECT INPUT-FILE-1  ASSIGN TO UTILITY.  SELECT
01210      SORT-FILE-1  ASSIGN TO UTILITY UNITS.  SELECT FILE-2
01220      ASSIGN TO UTILITY.  SELECT FILE-3  ASSIGN TO UTILITY.
01230
01240  DATA DIVISION.
01250  FILE SECTION.
01260  FD  INPUT-FILE-1  BLOCK CONTAINS 5 RECORDS
01270      VALUE OF FILE-ID 'F401'.
01280  01  INPUT-RECORD.
01290      02  FIELD-A  PICTURE X(20).
01300      02  FIELD-C  PICTURE 9(10).
01310      02  FIELD-D  PICTURE X(15).
01320      02  FILLER   PICTURE X(900).
01330      02  FIELD-B  PICTURE X(20).
01340      02  FIELD-E  PICTURE 9(5).
01350      02  FIELD-G  PICTURE X(25).
01360      02  FIELD-F  PICTURE 9(5).
(1)-01370  SD  SORT-FILE-1, VALUE OF FILE-ID IS 'SF1'.
   01380  01  SORT-RECORD.
   01390      02  FIELD-AA  PICTURE X(20).
   01410      02  FIELD-CC  PICTURE 9(10).
(2)01420      02  FIELD-BB  PICTURE X(20).
   01430      02  FIELD-DD  PICTURE X(15).
   01440      02  FIELD-EE  PICTURE 9(5).
   01450      02  FIELD-FF  PICTURE 9(5).
01460  FD  FILE-2  BLOCK CONTAINS 10 RECORDS
01470      VALUE OF FILE-ID 'F402'.
01480  01  FILE-2-RECORD.
01490      02  FIELD-EEE  PICTURE $$$$$9.
01500      02  FILLER-A   PICTURE X(2).
01510      02  FIELD-FFF  PICTURE 9(5).
01520      02  FILLER-B   PICTURE X(2).
01530      02  FIELD-AAA  PICTURE X(20).
01540      02  FIELD-BBB  PICTURE X(20).
01550  FD  FILE-3  BLOCK CONTAINS 15 RECORDS
01560      VALUE OF FILE-ID 'F403'.
01570  01  FILE-3-RECORD  PICTURE X(75).
```

FIGURE S-2. EXAMPLE OF COBOL SOURCE PROGRAM WITH SORT FEATURE.(Continued)

```
01580
01590  PROCEDURE DIVISION.
01591
01592      OPEN INPUT INPUT-FILE-1, OUTPUT FILE-2, FILE-3.
(3) 01593  SORT SORT-FILE-1  ASCENDING FIELD-AA  DESCENDING FIELD-BB,
01594      ASCENDING FIELD-CC  INPUT PROCEDURE RECORD-SELECTION  OUTPUT
01595      PROCEDURE PROCESS-SORTED-RECORDS.  CLOSE INPUT-FILE-1, FILE-2,
01596      FILE-3.  STOP RUN.
01597
01620  RECORD-SELECTION SECTION.
01630  PARAGRAPH-1.  READ INPUT-FILE-1  AT END GO TO PARAGRAPH-2.
01640      IF FIELD-A = FIELD-B GO TO PARAGRAPH-1  ELSE
(4) 01650  MOVE FIELD-A TO FIELD-AA   MOVE FIELD-F TO FIELD-FF
01660      MOVE FIELD-C TO FIELD-CC   MOVE FIELD-B TO FIELD-BB
01670      MOVE FIELD-D TO FIELD-DD   MOVE FIELD-E TO FIELD-EE
01680      RELEASE SORT-RECORD.  GO TO PARAGRAPH-1.
01690  PARAGRAPH-2.  EXIT.
01700  PROCESS-SORTED-RECORDS SECTION.
01710  PARAGRAPH-3.  RETURN SORT-FILE-1  AT END GO TO PARAGRAPH-4.
01720      IF FIELD-FF = FIELD-EE WRITE FILE-3-RECORD FROM
01730      SORT-RECORD GO TO PARAGRAPH-3  ELSE
(5) 01740  MOVE FIELD-EE TO FIELD-EEE   MOVE FIELD-FF TO FIELD-FFF
01750      MOVE FIELD-AA TO FIELD-AAA   MOVE FIELD-BB TO FIELD-BBB
01760      MOVE SPACES TO FILLER-A, FILLER-B  WRITE FILE-2-RECORD.
01770      GO TO PARAGRAPH-3.
01780  PARAGRAPH-4.  EXIT.
```

KEY

1 SORT DESCRIPTION ENTRY
2 RECORD DESCRIPTION ENTRY
3 MAIN-BODY OF SOURCE PROGRAM (excluding Sort input and output sections)
4 INPUT PROCEDURE
5 OUTPUT PROCEDURE

FIGURE S-2. EXAMPLE OF COBOL SOURCE PROGRAM WITH SORT FEATURE.(Continued)

SECTION 9: SOURCE PROGRAM LIBRARY FACILITY

Prewritten source program entries can be included in a COBOL program at compile time. Thus, an installation can utilize standard file descriptions, record descriptions, or procedures without having to repeat programming them. These entries and procedures are contained in a user-created library. They are included in a source program by means of a COPY clause or an INCLUDE statement.

COPY Clause

The COPY clause permits the user to include prewritten Data Division entries or Environment Division clauses in his source program. The COPY clause is written in the Data Division in one of the following forms:

Option 1
(Within the Input-Output section of the Environment Division)

SELECT file-name COPY library-name [FROM LIBRARY].

Option 2
(Within a File Description or Sort Description entry)

{FD / SD} file-name COPY library-name [FROM LIBRARY].

F Only Option 3
(Within a Saved Area Description entry)

SA saved-area-name COPY library-name [FROM LIBRARY].

Option 4
(Within a Record Description entry)

01 data-name COPY library-name [FROM LIBRARY].

F Only Option 5
(Within a Report Description Entry)

RD data-name [CODE non-numeric-literal]

COPY library-name [FROM LIBRARY]

F Only Option 6
(Within a Report Group Description entry)

01 [data-name] COPY library-name [FROM LIBRARY].

Option 7
(Within a 77 Record Description entry)

77 data-name COPY library-name [FROM LIBRARY].

Library-name is contained in the user's library and identifies the entries to be copied. It is an external-name and must follow the rules for external-name formation.

A COPY clause may be preceded by other information on a source program card, and may be written on more than one card; however on a given card, containing the completion of a COPY clause, there must be no information beyond the clause-terminating period. The material introduced into the source program by the COPY statement will follow the COPY statement on the listing, beginning on the next line.

When Option 1 is written, the clauses that are copied are those in the remainder of the Input-Output Section associated with the file identified by library-name.

When Options 2, 3, or 5 are written, then the file, sort, saved-area, or report description entry and its respective record or report groupand report element description entries are copied from the library.

When Options 4 or 6 are written, the entries that are copied from the library are those including and subordinate to the data-name with the 01 level number that begins the entries identified by library-name.

When Option 7 is written the entries that are copied from the library are the 88 level entries associated with the 77 level data-name identified by library-name.

In the Data Division and Environment Division of the source program, file-name or data-name or saved-area-name in a COPY clause will replace a corresponding file-name, data-name, or saved-area-name of the library entry. It is assumed that each library entry will contain whatever clauses are needed to complete the source program entry.

INCLUDE Statement

The INCLUDE statement permits the user to include prewritten procedures in the Procedure Division of his source program. The INCLUDE statement has the following formats:

Option 1 (For insertion of a paragraph):

paragraph-name. INCLUDE library-name

Option 2 (For insertion of a section):

section-name SECTION. INCLUDE library-name

Library-name is contained in the user's library and identifies the entries to be copied. It is an external name and must follow the rules for external name formation.

An INCLUDE statement may be preceded by other information on a source card, and may be written on more than one card; however on a given card, containing the completion of an INCLUDE statement, there must be no information beyond the clause-terminating period. The material from the library will follow the INCLUDE statement on the listing.

The library entries for paragraphs and sections must not contain INCLUDE statements.

In the source program, the library entry is included in its entirety, including the paragraph-name or section-name of the library entry. No substitution of names, within the library entry, is performed. The library entry is included in its entirety.

A complete program may be included as an entry in the user's library, and may be used as the basis of compilation. When this is the case, input to the compiler is a BASIS card, followed by any number of INSERT and/or DELETE cards, and followed by any number of debugging packets, if desired. Debugging packets are described in Section 9.

The format of the BASIS card is:

```
1       8
-----------------
BASIS   library-name
```

Library-name is an external name: it names the complete program entry used as a basis for the compilation.

If INSERT or DELETE cards follow the BASIS card, the library entry is modified prior to being processed by the compiler.

The format of the INSERT card is:

```
1       8
--------------------------
INSERT  sequence-number-field
```

The format of the DELETE card is:

```
1       8-72
--------------------
DELETE  sequence-number-field
```

A sequence-number-field consists of entries of the form a-b, or c separated by comma and a single blank, where a, b, and c are individual sequence-numbers of the basic library entry (appearing in columns 1-6 of the basic library entry).

At least one new source program card must follow an INSERT card, for insertion after the card specified by the sequence-number-field.

Source program cards may follow a DELETE card, for insertion before the card following the last one deleted.

STERLING CURRENCY FEATURE

Ext

System/360 COBOL provides facilities for handling sterling currency items by means of an expansion of the PICTURE clause. Additional options and formats, necessitated by the non-decimal nature of sterling, and by the conventions by which sterling amounts are represented in punched cards, are also available.

Sterling amounts are normally expressed in pounds, shillings and pence, in that order. There are twenty shillings in a pound, and twelve pence in a shilling. Though sterling amounts are sometimes expressed in shillings and pence only (in which case the number of shillings may exceed 99), within machine systems shillings will always be expressed as a two-digit field. Pence, when in the form of integers, likewise will be expressed as a two-digit field. However, provision must be made for pence to be expressed as decimal fractions as well, as in the form 17s. 10.237d.

The IBM method for representing sterling amounts in punched cards uses two columns for shillings and one for pence. Tenpence (10d.) is represented by an '11' punch and elevenpence (11d.) by a '12' punch. The British Standards Institution (B.S.I.) representation uses single columns for both shillings and pence. B.S.I. pence representation for tenpence and elevenpence is the reverse of that of IBM: an '11' punch is used for 11d. and a '12' punch for 10d. B.S.I. representation for shillings consists of a '12' punch for 10 shillings and double punches A to I for eleven to nineteen shillings.

The indicated representations may be used separately or in combination, resulting in four possible conventions.

1. IBM pence - IBM shillings

2. IBM pence - B.S.I. shillings

3. B.S.I. pence - B.S.I. shillings

4. B.S.I. pence - IBM shillings

Any of these conventions may be associated with any number of digits, or no digits, in the pound field; and any number of decimals, or no decimals, of pence. In addition, sign representation may be present as an overpunch in one of several allowable positions in the amount, or may be separately entered from another field.

Two formats are provided by System/360 COBOL in the PICTURE clause for the representaion of sterling amounts, Sterling Non-Report forma and Sterling Report format. In the formats that follow, n stands for a positive nonzero integer. When such an integer is used, it must be parenthesized. The characters 6 7 8 9 B Z V . : s d CR - are the PICTURE characters used to describe Sterling items.

The format for the Sterling non-report PICTURE is:

PICTURE IS 9 [(n)] [V] [8] 8 [V] {6 [6] / 7 [7]} [[V] 9 (n)]

[USAGE IS DISPLAY]

The representation for pounds is 9(n) [V] where:

a. The character 9 indicates that a character position will always contain a numeric character, and may extend to n positions.

b. The character V indicates the position of an assumed pound separator. In all other respects it is identical to V when used to indicate an assumed decimal point. As each sterling field is defined by the use of a different character, the use of V for this purpose, in this format, is optional.

c. An entry must always appear in the pound field for a picture to be valid.

The representation for shillings is [8] 8 [V] where:

a. The characters [8] 8 indicate the position of the shilling field, and the convention by which shillings are represented in punched cards. 88 indicates IBM shilling representation occupying a two column field. 8 indicates B.S.I. single column shilling representation. An entry must always appear in the shilling field for a picture to be valid.

b. The character V indicates the position of an assumed shilling separator. In all other respects it is identical to V when used to indicate an assumed decimal point.

The representation for pence is {6 [6] / 7 [7]} [[V] 9 (n)]

a. The character 6 indicates IBM single column pence representation wherein 10d. is represented by an '11' punch and 11d. by a '12' punch. The characters 66 indicate two column representation of pence, usually from some external medium other than punched cards.

b. The character 7 indicates B.S.I. single column pence representation wherein 10d. is represented by a '12' punch and 11d. by an '11' punch. The characters 77 indicate two column representation of pence. Consequently, 66 and 77 serve the same purpose and are interchangeable. An entry must always appear in the pence field for a picture to be valid.

c. The character V indicates the position of an assumed decimal point in the pence field. Its properties and use are identical with that of V in dollar amounts. Decimal positions in the pence field may extend to n positions.

STERLING SIGN REPRESENTATION

Signs for sterling amounts may be entered as overpunches in one of

several allowable positions of the amount. A sign is indicated by an embedded S in the non-report picture immediately to the left of the position containing the overpunch. Allowable overpunch positions are the high order and low order positions of the pound field, the high order shilling digit in two column shilling representation, the low order pence digit in two column pence representation, or the least significant decimal position of pence. Examples of such a picture are:

9S98V6V99

9 (3) 8V6S6V9

STERLING REPORT

The format for the Sterling report PICTURE is:

PICTURE IS

```
[ pound-report-string { B (n)                    } ]  { 99  }  { B (n)                    }
                      { . (n)   [B (n)]          }    { ZZ  }  { . (n)   [B (n)]          }
                      { : [B (n)]                }    { Z8  }  { : [B (n)]                }
                      { /                        }             { /                        }
                      { B/B                      }             { B/B                      }
                      { V                        }             { V                        }
                      { s [.]  ] B (n)]          }             { s [.]  ] B (n)]          }

{ 99 }  [ [d]   [.]      ]   [ [B (n)] { CR } ]
{ Z9 }  [ 9 (n) [d]  [.] ]   [       { -  } ]
{ ZZ }
{ Z8 }
```

USAGE IS DISPLAY-ST

The picture for STERLING REPORT is composed of a sequence of characters representing the fields for, respectively, (i) pounds (ii) pound separators (iii) shillings (iv) shilling separators (v) pence integers (vi) pence decimal fractions and pence terminators. The USAGE IS DISPLAY-ST clause is necessary to enable the compiler to distinguish between sterling and decimal data in cases where their formats may be the same.

1. a. Pound-report-string for the representation of pounds is similar to the report-form option for decimal fields. The editing characters that may be combined to describe a pound report item are: 9 Z * , B £ + -. With the exception of the pound sign (£) the editing characters have the same meaning in pound-report-string as the report form for decimal fields.

   With one exception, the pound sign may be equated to the dollar sign in terms of function and manner of use. Specifically, the pound sign may be used as a floating string character and, as with the dollar sign, may be floated through Bs and/or 0s and/or commas, should they be embedded in the floating string. Examples of such strings are:

£,£££,£99

££B£££9

b. The exception to equating the pound and dollar signs consists of an option available in pound-report-strings. With this option the floating pound sign may be suppressed when the value of the pound field is zero. Such suppression is the pound-report-string equivalent of the BLANK WHEN ZERO clause. The floating pound string may be described in either of two formats:

	Format	Values of pounds field	Output
1)	£. . .£	1	b£1
		0	bb£
2)	£(n)	1	b£1
		0	bbb

The use of parenthesis to indicate multiplicity of pound signs in a picture, as in format 2, indicates the pound sign suppression option is desired.

c. The single character £ indicates the position containing a fixed pound sign.

2. The representation for pound separators is:

$$\left\{\begin{array}{l} B(n) \\ .(n) \quad [B(n)] \\ :[B(n)] \\ / \\ B/B \\ V \end{array}\right\}$$

a. The characters B(n) specify n character positions in which blanks will be inserted.

b. The characters :[B(n)] specify a colon which may be followed by n character positions containing blanks.

c. The characters .(n) [B(n)] specify n character positions containing periods which may be followed by n character positions containing blanks.

d. The character / may stand alone as a separator, or it may be preceded by one blank and followed by another.

e. The character V must be used if no other pound separator is specified. Its use is internal only and enables the compiler to properly align pound and shilling fields.

f. In cases where the pound sign suppression option has been specified, and the pound field is equal to zero, suppression will be extended to include the pound separator. Stated generally, under the pound sign suppression option the suppression of all digits to the left of a separator will result in the suppression of the separator as well. Consequently, an amount equal to zero will result in output consisting only of spaces.

3. The representation for shillings is:

$$\left\{\begin{array}{l} 99 \\ ZZ \\ Z8 \end{array}\right\}$$

The character 8 is identical to Z when all digits to the left, including those in the pound field, have been suppressed, and is identical to 9 if they have not been.

4. The representation for shilling separators is:

```
{ B (n)
{ . (n) [B (n) ]
{ :  [B (n) ]
{ /
{ B/B
{ s [.] [B (n) ]
{ V
```

a. Those characters which are used also as pound separators are discussed under that heading.
b. The characters s or s. may stand alone as separators, followed immediately by the high order pence digit, or they may be followed by n spaces, when written in the formats sB (n) and s.B (n) .
c. The character V must be used if no other shilling separator is specified. Its use is internal only and enables the compiler to properly align pounds and shillings.

5. The representation for pence integers is:

```
{ 99
{ Z9
{ ZZ
{ Z8
```

a. The character 8 is identical to Z when all digits to the left, including those in the pound and shilling fields, have been suppressed, and is identical to 9 when they have not been. The remaining characters have been explained under other headings. If there are no positions of pence decimals, the pence terminator follows immediately. If there are decimals, the terminators follow the low order decimal position.

6. The representation for pence decimal fractions and terminators is:

```
[ [d] [.]          ][ B (n) {CR} ]
[ 9 (n)  [d]  [.]  ][       {-  } ]
```

a. The upper set of characters d and . represents pence terminators which may be used if there is no pence decimal fraction. The characters 9 (n) indicate the number of positions the pence decimal fraction will occupy, should there be one. The second set of characters d and. indicates that these pence terminators may also be used <u>following</u> a pence decimal fraction.

b. The characters B (n) CR and - indicate terminators which may follow those discussed above. The characters CR or - may append to an amount to indicate a debit and may immediately follow a previous pence terminator or low order pence digit, or they may be preceded by <u>n</u> spaces. The + sign is not used in sterling amounts.

Both COBOL E and COBOL F allow the user to employ sterling non-report items as operands in connection with other numeric operands in MOVE, ADD, and SUBTRACT statements.

For COBOL F, use of sterling non-report items in other arithmetic statements is also permitted. Only use as an exponent in an arithmetic statement is prohibited. Sterling report-items are used for editing sterling amounts.

Decimal items moved to sterling report and sterling non-report are considered as pence.

INTERNATIONAL CONSIDERATIONS

1. Installations may interchange the function of the comma and decimal point characters in numeric literals and the PICTURE clause.

2. Installations may alter the compiler's character set for non-English language requirements, so that, for example, data-names may be composed of letters of a particular national alphabet.

3. The PICTURE of report items may terminate with the currency symbol in cases where the graphic $ is supplanted by a particular national currency symbol.

4. Sentences may be substituted to allow translation (by modification) of output messages into any non-English language.

The following statements are provided for program debugging. They may appear anywhre in a System/360 COBOL program or in a compile-time debugging packet.

TRACE

The format of the TRACE statement is:

```
{READY}
{RESET} TRACE
```

After a READY TRACE statement is executed, each time execution of a paragraph or section begins, a message is written of arrival at such a point. The message is written on the system logical output device (SYSOUT).

The execution of a RESET TRACE statement terminates the functions of a previous READY TRACE statement.

EXHIBIT

The format of the EXHIBIT statement is:

```
                 {NAMED          }  {data-name        }
EXHIBIT          {CHANGED NAMED  }  {non-numeric-literal}  ...
                 {CHANGED        }
```

The execution of an EXHIBIT NAMED statement causes a formatted display of the data-names (or non-numeric literals) listed in the statement. The system logical output device (SYSOUT) is used. The format of the output for each data-name listed in the NAMED or CHANGED NAMED form of an EXHIBIT statement is:

blank
original data-name (including qualifiers, if written)
blank
equal sign
blank
value of data-name

Literals listed in the statement are preceded by a blank, when displayed.

The CHANGED form of the EXHIBIT statement provides for a display of items when they change value, compared to the value at the previous time the EXHIBIT CHANGED statement was executed. The initial time such a statement is executed, all values are considered changed; they are displayed and saved for purposes of comparison.

Note that, if two distinct EXHIBIT CHANGED data-name statements appear in a program, changes in data-name are associated with the two separate statements. Depending on the path of program flow, the values of data-name saved for comparison may differ for the two statements.

If the list of operands in an EXHIBIT CHANGED statement includes literals, they are printed as remarks and are preceeded by a blank.

For COBOL E, only one data-name may be listed in an EXHIBIT CHANGED statement.

F Only If there are two or more data-names as operands of EXHIBIT CHANGED, and some but not all are changed from the previous execution of the statement, only the changed values are displayed. The positions reserved for a given operand in the data to be displayed are blank when the value of the operand is not changed. The programmer can thus create a fixed columnar format for the data to be displayed by use of the EXHIBIT CHANGED.

The CHANGED NAMED form of the EXHIBIT statement causes a printout of each changed value for items listed in the statement. Only those values representing changes and their identifying names are printed. A fixed columnar format for the data to be displayed cannot be created with EXHIBIT CHANGED NAMED.

ON (Count-Conditional Statement)

The format of the ON statement is:

```
ON integer-1 [AND EVERY integer-2] [UNTIL integer-3]

    {imperative-statement...}
    {NEXT SENTENCE          }

      [{ELSE     }{statement ...  }]
      [{OTHERWISE}{NEXT SENTENCE}]
```

The ON statement is a conditional statement. It specifies when the statements it contains are to be executed. ELSE (OR OTHERWISE) NEXT SENTENCE may be omitted if it immediately precedes the period for the sentence.

The count-condition (integer-1 AND EVERY integer-2 UNTIL integer-3) is evaluated as follows:

Each ON statement has a compiler-generated counter associated with it. The counter is initialized in the object program with a value of zero.

Each time the path of program flow reaches the ON statement, the counter is advanced by 1. Where K is any positive integer, if the value of the counter is equal to integer-1 + (K*integer-2), but is less than integer-3 if specified, then the imperative statements (or NEXT SENTENCE) are executed. Otherwise, the statements after ELSE (or NEXT SENTENCE) are executed. If the ELSE option does not appear, the next sentence is executed.

If integer-2 is not given, it is assumed that integer-2 has a value of 1. If integer-3 is not given, no upper limit is assumed for it.

If neither integer-2 nor integer-3 is specified, the imperative statements are executed only once.

Examples:

ON 2 AND EVERY 2 UNTIL 10 DISPLAY A ELSE DISPLAY B.

On the second, fourth, sixth, and eighth times, A is displayed. B is displayed at all other times.

ON 3 DISPLAY A.

On the third time through the count-conditional statement, A is displayed. No action is taken at any other time.

COMPILE-TIME DEBUGGING PACKET

Debugging statements for a given paragraph or section in a program may be grouped together into a debugging packet. These statements will be compiled with the source language program, and will be executed at object time. Each packet refers to a specified paragraph-name or section-name in the Procedure Division. Compile-time debugging packets are grouped together and are placed immediately following the last card of the source program.

Each compile-time debug packet is headed by the control card *DEBUG. The general form of this card is

```
1       8
*DEBUG location [,TRY]
```

where the parameters are described as follows:

location This is the COBOL section-name or paragraph-name (qualified, if necessary) indicating the point in the program at which the packet is to be executed. Effectively, the statements in the packet are executed as if they were physically placed in the source program following the section-name or paragraph-name, but preceding the text associated with the name.

TRY If this option is used, immediate loading and execution of the object program will be allowed even if an error appears within a debug packet. If TRY is not specified a source program error, when encountered in the procedural text of a debug packet, will prevent loading and execution of the object program.

A debug packet may consist of any procedural statements conforming to the requirements of System/360 COBOL. A GO TO, PERFORM, or ALTER statement in a debug packet may refer to a procedure-name in any debug packet or in the main body of the Procedure Division.

APPENDIX A: GLOSSARY OF LOWER-CASE WORDS IN COBOL FORMATS

In this appendix, definitions are provided for certain elements of the COBOL language. These definitions are presented in a uniform system of notation, explained in the following paragraphs. This notation is useful in describing the COBOL language, although it is not part of COBOL. All definitions presented in this notation are syntactical definitions. They define the structure, rather than the meaning, of the defined element.

1. All words printed entirely in capital letters are reserved words. These are words that have preassigned meanings in the COBOL language. Words in capital letters represent an actual occurrence of those words.

Example: ADD

When this is specified, the letters ADD are indicated.

2. All punctuation and special characters (except those symbols described in the following paragraphs) represent the actual occurrence of those characters. Punctuation is essential where it is shown. Additional punctuation can be inserted, according to the rules for punctuation specified in this publication.

3. Lower-case words that appear in a definition are themselves defined under the proper entry.

Example: integer

This specifies the appearance of an item of the class "integer."

4. | (the or sign) The or sign indicates a choice.

Example: +|-

This specifies a plus sign or a minus sign.

5. ::= (the defined-as symbol) This symbol means "defined as."

Example: digit ::=1|2|3|4|5|6|7|8|9|0

6. . . . (the ellipsis) The ellipsis indicates that the immediately preceding unit may occur one or more times in succession. A unit, in this and succeeding paragraphs, means either a single reserved word, a single lower-case word, or a group of lower-case words and one or more reserved words enclosed in brackets or braces. If an item is enclosed in brackets or braces, the entire unit of which it is a part must be repeated if repetition is specified.

Example: unsigned-integer::=digit. . .

This defines an unsigned integer as any number of sequential digits.

7. [ ] (brackets) Brackets are used to indicate that an item is optional. The programmer may use the item or not, depending on the requirements of his program.

Example: non-negative-integer::= [+] digit. . .

This defines a non-negative integer as a series of digits optionally preceded by a plus sign.

8. When brackets surround items separated by the or sign, any one of the items enclosed, or none of them, may be chosen.

Example: integral-number::= [+|-] digit. . .

This specifies that an integral number is a series of digits preceded by a minus sign, a plus sign, or neither.

9. { } (braces) When braces surround items separated by the or sign, one of the items shown must be chosen.

Example: signed-integer::= {+|-}digit. . .

This defines a signed integer as a series of digits that must be preceded by either a plus sign or a minus sign.

10. Braces may also be used to indicate that the enclosed items are to be treated as a unit.

Example: [letter] {digit letter} . . .

This specifies a sequence of any length consisting of digits alternating with letters.

11. not (the not symbol) This symbol indicates that the unit following it may not occur.

Example: digit not 0

This specifies any member of the class digit except 0. Combined with the definition of digit used above, it specifies the equivalent of

1|2|3|4|5|6|7|8|9

12. min (the minimum symbol) This indicates the minimum number of times that a unit may occur. When it is used without an accompanying maximum symbol (defined below), the implied maximum of times the unit may occur is infinity.

Example: min 3 digit

This specifies a sequence of no less than three digits. There is no upper limit on the length of this sequence.

Example: min 5{digit|letter}

This specifies a sequence of at least five letters and digits, in any order.

13. max (the maximum symbol) This specifies the maximum number of times that the following unit may appear in succession. When max appears without an associated min, a minimum of zero occurrences is assumed.

Example: max 18 digit

This specifies that no digits, one digit, or a sequence of up to 18 digits may be indicated at this point by the programmer.

Example: min 2 max 6{digit|letter}

This specifies that digits and letters, intermixed in any succession, must occur in a sequence at least two long, or at most six long.

Example: min 7 max 7 digit

This specifies a sequence of exactly seven digits.

14. blank This symbol represents the occurence of a space.

Example: {.|blank}

This specifies that either a period or a space must occur at this point.

15. Further restrictions and explanations will be found in the text.

16. Certain special restrictions and definitions that apply only to the Sterling Currency Feature are not included in this glossary. They are discussed in Section 10.

alphabetic-literal::=	'min 1 max 120{letter|blank|-}
alpha-form::=	min 1 max 30 A[(integer)]

The sum of the value of integer plus the number of As must not exceed 30.

alphanumeric-literal::=	'min 1 max 120{character not ' } '
an-form::=	min 1 max 30 X[(integer)]

The sum of the value of integer plus the number of Xs must not exceed 30.

argument::=	data-name|file-name|procedure-name
arithmetic expression::=	[( ] {{{numeric-literal|floating-point-literal|data-name|arithmetic-expression}arithmetic-operator}|-} [numeric-literal|floating-point-literal|data-name|arithmetic-expression} [ )]

The use of the) is obligatory if the (has been used. Data-names used in arithmetic-expressions must refer to numeric data.

arithmetic-operator::=	+|-|*|/|**
character::=	COBOL-character| {EBCDIC-character not COBOL-character}
clause::=	Individual clauses are described in appropriate sections of the text.
COBOL-character::=	letter |digit|+|-|/|*|=|$|,| (|) |.|;|<|>|blank
comment::=	COBOL-character . . .{ . blank}
compound-condition::=	[NOT] {test-condition{AND/OR} test-condition}
compiler-directing-statement::=	See "Compiler-Directing Statements" in text.
condition::=	test-condition|event-condition
conditional-statement::=	See "Conditional Statements" in text.
data-name::=	{[{digit|letter}max n{digit|-|letter}] letter [max m{digit|-|

letter}{digit|letter}]}not reserved-word

The sum of m plus n is 28.

device-number::= unit-record-device|utility-device|direct-access-device

digit::= 0|1|2|3|4|5|6|7|8|9

direct-access-device::= 1302|7320|2301|2311|2321

division-name::= DATA|PROCEDURE|ENVIRONMENT|IDENTIFICATION

EBCDIC-character Any character in the IBM Extended Binary Coded Decimal Interchange Character set.

elementary item::= alphabetic-item|alphanumeric-item|report-item|external-decimal-item|internal-decimal-item|binary-item|external-floating-point-item|internal-floating-point-item

entry-name::= external-name

event-condition::= {[AT] END}|{INVALID KEY}|{[ON] SIZE ERROR}

exponent::= min 1 max 2 digit

external-name::= 'letter max 7{digit|letter}'

figurative-constant::= {HIGH-VALUE|HIGH-VALUES}|{LOW-VALUE|LOW-VALUES}|{QUOTE|QUOTES}|{SPACE|SPACES}|ALL'{character not '}'|{ZERO | ZEROES | ZEROS}

file-name::= data-name

floating-point-literal::= [+|-]mantissa E[+|-] exponent

fp-form::= {+|-} max n 9[(integer-1)] [V|.]min 1 min|max m 9[(integer-2)] E{+|-}99

The sum of m plus n plus the values of integer-1 plus integer-2 must not exceed 16.

imperative-statement::= See "Imperative Statements" in text.

integer::= [max n digit] digit not 0 [max m digit]

The sum of m plus n is 17.

letter::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z

level-indicator::= FD|SD|SA|RD|level-number

level-number::= {{0|1|2|3|4}digit}|77|88

library-name::= external-name

literal::= numeric-literal|non-numeric-literal|floating-point-literal

logical operator::= AND|OR|NOT

mantissa::= max m digit . digit max n digit

The sum of m plus n is 15.

model-number::= The model number of an IBM 360/ series computer.

name::= data-name|procedure-name|external-name|program-name

non-numeric-literal::= alphabetic-literal|alphanumeric-literal

numeric-form::= {min 1 9[P...] [V]}|
{[V] [P...] min 1 9}|
{9... [V] 9...}

The form (integer) placed after a 9 or a P specifies the equivalent of the appearance of that character integer times. The sum of all 9s and Ps plus the value of all integers must not exceed 18.

numeric-literal::= [+|-] max m digit [.] digit max n digit

The sum of m plus n is 17.

operand::= literal|figurative-constant|data-name|arithmetic-expression

paragraph::= [paragraph-name.] sentence. . .

paragraph-name::= I-O-CONTROL|FILE-CONTROL|SOURCE-COMPUTER|OBJECT-COMPUTER|procedure-name

procedure-name::= {{digit|letter} [max 28 [digit|letter|-] {digit|letter}]}not reserved-word

program-name::= letter max 7{digit|letter}

qualifier::= {OF|IN} {data-name|section-name}

record-name::= data-name

relational-operator::= {GREATER [THAN] >}|{EQUAL TO|=}|
{LESS [THAN] <}

report-form::= { +|-|$ |[*|Z]| P|,|B|0 | 9|[V|.]|
[CR|DB]} . . .

The valid combinations of these characters are described under "Report-Form Option." The form (integer) placed after any of these characters except V . CR and DB specifies the equivalent of the appearance of that character integer times.

report-name::= data-name

reserved-word::= Any word in the System/360 COBOL Word List, Appendix B in this publication.

```
saved-area-name::=              data-name

section::=                      section-header paragraph . . .

section-header::=               section-name SECTION.

section-name::=                 CONFIGURATION|INPUT-OUTPUT|FILE
                                |WORKING-STORAGE|LINKAGE|procedure-name

sentence::=                     {statement...|USE-sentence|
                                SELECT-sentence}.

simple-condition::=             [NOT] {relation-test|sign-test
                                |class-test|condition-name-test}

sort-file-name::=               data-name

statement::=                    imperative-statement|conditional-statement
                                |compiler-directing-statement
```

These types of statements are defined in the appropriate portions of the text.

```
test-condition::=               simple-condition|compound-condition

unit-record-device::=           1402|1403|1404|1442|1443|1445|2201

utility-device::=               1302|2400|2301|2311|2321|7320|7340

word::=                         name|reserved-word
```

APPENDIX B: SYSTEM/360 COBOL WORD LIST

The words listed below make up the complete System/360 COBOL vocabulary of reserved words.

ACCEPT
ACCESS
ACTUAL
ADD
ADVANCING
AFTER
ALL
ALPHABETIC
ALTER
ALTERNATE
AND
APPLY
ARE
AREA
AREAS
ASCENDING
ASSIGN
AT
AUTHOR

BEFORE
BEGINNING
BLANK
BLOCK
BY

CALL
CF
CH
CHANGED
CHARACTERS
CHECKING
CLOCK-UNITS
CLOSE
COBOL
CODE
COLUMN
COMPUTATIONAL
COMPUTATIONAL-1
COMPUTATIONAL-2
COMPUTATIONAL-3
COMPUTE
CONFIGURATION
CONSOLE
CONTAINS
CONTROL
CONTROLS
COPY
CORRESPONDING
COUNT
CREATING
CYCLES

DATA
DATE-COMPILED
DATE-WRITTEN
DE
DECLARATIVES
DEPENDING
DESCENDING
DETAIL
DIRECT-ACCESS
DISPLAY
DISPLAY-ST
DIVIDE
DIVISION

ELSE
END
ENDING
ENTER
ENTRY
ENVIRONMENT
EQUAL
ERROR
EVERY
EXAMINE
EXHIBIT
EXIT

FD
FILE
FILES
FILE-CONTROL
FILE-ID
FILLER
FINAL
FIRST
FOOTING
FOR
FORM-OVERFLOW
FROM

GENERATE
GIVING
GO
GREATER
GROUP

HEADING
HIGH-VALUE
HIGH-VALUES
HOLD

IBM-360
IDENTIFICATION
IF
IN
INCLUDE
INDEXED
INDICATE
INITIATE
INPUT
INPUT-OUTPUT
INSTALLATION
INTO
INVALID
I-O
I-O-CONTROL
IS

JUSTIFIED

KEY

LABEL
LABELS
LAST
LEADING
LEFT
LESS
LIBRARY
LINE-COUNTER

LINE
LINES
LINKAGE
LOCK
LOW-VALUE
LOW-VALUES

MORE-LABELS
MOVE
MULTIPLY

NAMED
NEGATIVE
NEXT
NO
NOT
NOTE
NUMERIC

OBJECT-COMPUTER
OCCURS
OF
OH
OMITTED
ON
OPEN
OR
ORGANIZATION
OTHERWISE
OUTPUT
OV
OVERFLOW

PAGE
PAGE-COUNTER
PERFORM
PF
PH
PICTURE
PLUS
POSITIVE
PRINT-SWITCH
PROCEDURE
PROGRAM-ID
PROCEED
PROCESS
PROCESSING
PROTECTION

QUOTE
QUOTES

RANDOM
RD
READ
READY
RECORD
RECORDS
RELATIVE
RELEASE
REMARKS
REPLACING
REPORT
REPORTING
REPORTS
RERUN
RESERVE
RESET
RESTRICTED
RETURN
REVERSED
REWIND
REWRITE
RF
RH
RIGHT
ROUNDED
RUN

SA
SAME
SD
SEARCH
SECTION
SECURITY
SELECT
SENTENCE
SEQUENTIAL
SIZE
SORT
SOURCE
SOURCE-COMPUTER
SPACE
SPACES
STOP
SUBTRACT
SUM
SYMBOLIC
SYNCHRONIZED
SYSPCH

TALLY
TALLYING
TERMINATE
THAN
THEN
THRU
TIMES
TO
TRACE
TRACKS
TRANSFORM
TYPE

UNIT
UNIT-RECORD
UNITS
UNTIL
UPON
USE
USING
UTILITY

VALUE
VARYING
VIA

WHEN
WITH
WITHOUT
WORKING-STORAGE
WRITE

ZERO
ZEROES
ZEROS

APPENDIX C: ASYNCHRONOUS PROCESSING SAMPLE PROGRAM

This program (RANDOM) is an example of an asynchronous processing. It is a simple banking example in which transactions (deposits and withdrawals) are read in from tape and processed asynchronously against a direct-access accounts file with relative data organization.

Two files and a saved area (SA) are used by the program. One file,the transactions file TRANSACT is a tape file with 80 character, card image records blocked 5 to a physical record. Each record contains a ten-digit "account-number" in columns 11 to 20, a one digit "code" indicating deposits (D) or withdrawal (W) in column 26, and an eight-digit "amount" in columns 32 to 39. The last two digits are assumed to be the cents portion of the amount. All eight digits of the "amount" field, including leading zeros, must be punched.

The second file is the master-file ACCOUNTS. It is an asynchronously processed direct access file with relative data organization and is contained on three direct-access units. Its symbolic key is SA-ACCOUNT-NUMBER and its actual key is SA-TRACK-NUMBER. Both of these fields are in the saved area. More than one unit is used in order to provide greater asynchronous processing. With only one unit, true asynchronous processing is not possible because the device must complete one operation before the next can be initiated and, therefore, the records will be read and processed in the order in which they are accessed. With multiple units, greater asynchronous processing can be obtained because records which are on different units can be accessed simultaneously and the record which is found first can be processed first. The file ACCOUNTS contains twenty-two character unblocked records. Each record contains an assigned eight-digit internal decimal balance field (ACC-BALANCE), a twelve-character name field (ACC-NAME) and a sixty- character address field (ACC-ADDRESS).

The saved area TRANSFER AREA is used to pass data from the main body (in-line procedure) of the program to the asynchronous processing (out-of-line) procedure. It contains one record (SA-REC) consisting of a transaction area (SA-TRANS-IN) and a work area (SA-W-S). The transaction area contains an account-number field, a code field, and an amount field, corresponding to the fields in the records in TRANSACT. It is used to pass a record from TRANSACT to the out- of-line procedure. The work area is used by the out-of-line procedure as temporary storage during its operation. The work area contains three fields, SA-OLD-BALANCE, SA-NEW-BALANCE, and SA-ERR-NO used by the out-of-line procedure to produce error messages, and a fourth field, SA-TRACK-NUMBER, which is the actual key of the ACCOUNTS file. The value for this field is calculated by the out-of-line procedure and is used when reading in order to locate the desired record in the direct-access file.

RANDOM is divided into two main segments, the in-line and the out-of-line procedures. The in-line procedure (the main body of the Procedure Division) opens the transactions file TRANSACT, and the accounts file ACCOUNTS, reads a transaction record, moves it into the saved area (TRANSFER-AREA), and initiates the out-of-line procedure (by means of a PROCESS statement). It then reads the next transaction

record. When an end of file is reached on the transaction file, the in-line procedure waits (by means of a HOLD statement) for the completion of any out-of-line procedures which are still active; it then closes the files (TRANSACT and ACCOUNTS), and stops.

The RANDOM UPDATE Section is the out-of-line procedure in the Declaratives portion of the Procedure Division. This procedure calculates the actual key for this record from the account number in the saved area, reads the record from ACCOUNTS, updates the record, and writes it back by means of a REWRITE statement onto the file. An out-of-line HOLD is also provided to force completion of the out-of-line cycles in the order in which they were initiated.

RANDOM provides for eight out-of-line cycles to be asynchronously processed with the in-line procedure. Each of these cycles is provided with a copy of the saved area and the input area of the ACCOUNTS file. When RANDOM is started, the in-line procedure is entered. The in-line procedure performs its functions and effectively runs continously until it executes the HOLD statement, independent of the out-of- line cycles. During the in-line processing, the PROCESS statement is executed many times. At each execution, a new out-of-line cycle also operates independently of the in-line procedure and any other out-of-line cycles.

Although the in-line procedure and the out-of-line cycles operate as though they are continuous and independent, operation is actually interleaved. The in-line procedure or an out-of-line cycle retains control until it has to wait for the completion of an input/output operation or, because of a HOLD statement, until it has to wait for another cycle. Control is then passed to another cycle or to the in-line procedure that is not in a wait status. The in-line procedure or an out-of-line cycle that gives up control will be resumed at some later time after the condition it is waiting for is satisfied.

```
IDENTIFICATION DIVISION.
  PROGRAM-ID.  RANDOM.
REMARKS.
    THIS PROGRAM IS AN EXAMPLE OF AN ASYNCHRONOUS
    PROCESSING PROGRAM .  IT IS A SIMPLE BANKING PROBLEM IN WHICH
    TRANSACTIONS (DEPOSITS AND WITHDRAWALS) ARE ASYNCHRONOUSLY
    PROCESSED AGAINST A 'RELATIVE' ORGANIZED DIRECT-ACCESS
    MASTER FILE.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
  SOURCE-COMPUTER. IBM-360.
  OBJECT-COMPUTER. IBM-360.
INPUT-OUTPUT SECTION.
  FILE-CONTROL.
    SELECT TRANSACT ASSIGN TO UTILITY.
    SELECT ACCOUNTS ASSIGN TO DIRECT-ACCESS UNITS
        ACCESS IS RANDOM
        ORGANIZATION IS RELATIVE
        SYMBOLIC KEY IS SA-ACCOUNT-NUMBER
        ACTUAL KEY IS SA-TRACK-NUMBER.
  I-O-CONTROL.
    APPLY RANDOM-UPDATE TO TRANSFER-AREA, ACCOUNTS FOR 8 CYCLES.

DATA DIVISION.
FILE SECTION.
FD  TRANSACT
        BLOCK CONTAINS 5 RECORDS
        RECORD CONTAINS 80 CHARACTERS
        VALUE OF FILE-ID 'E403'.
  01 TRANSACTION-RECORD.
    02 FILLER           PICTURE X(10).
    02 ACCOUNT-NUMBER   PICTURE X(10).
    02 FILLER           PICTURE X(5).
    02 TRANSACTION-CODE PICTURE X.
    02 FILLER           PICTURE X(5).
    02 AMOUNT           PICTURE 9(6)V99.
    02 FILLER           PICTURE X(41).
FD  ACCOUNTS
    RECORD CONTAINS 77 CHARACTERS
        VALUE OF FILE-ID 'E404'.
  01 ACCOUNT-RECORD.
    02 ACC-BALANCE  PICTURE S9(7)V99 COMPUTATIONAL-3.
    02 ACC-NAME.
        03 LAST         PICTURE X(10).
        03 FIRST-INIT   PICTURE X.
        03 MIDDLE-INIT PICTURE X.
    02 ACC-ADDRESS.
        03 STREET   PICTURE X(20).
        03 CITY     PICTURE X(20).
        03 STATE    PICTURE X(20).
SA  TRANSFER-AREA.
  01 SA-REC.
    02 SA-TRANS-IN.
        03 ACCOUNT-NUMBER PICTURE X(10).
```

```
           03 SA-ACCOUNT-NUMBER REDEFINES ACCOUNT-NUMBER OF SA-REC.
              04 SA-AN1 PICTURE 9(4).
              04 SA-AN2 PICTURE 9(4).
              04 SA-AN3 PICTURE 9(2).
           03 TRANSACTION-CODE PICTURE X.
               88 DEPOSIT    VALUE  'D'.
               88 WITHDRAWAL VALUE  'W'.
           03 AMOUNT PICTURE 9(6)V99.
       02 SA-W-S.
           03 SA-TRACK-NUMBER PICTURE S999 COMPUTATIONAL.
           03 SA-ERR-NO PICTURE 9.
           03 SA-WORK PICTURE S9999 COMPUTATIONAL.
           03 SA-OLD-BALANCE  PICTURE $*,***,**9.99-.
           03 SA-NEW-BALANCE  PICTURE $*,***,**9.99-.

PROCEDURE DIVISION.

DECLARATIVES.
RANDOM-UPDATE SECTION.
    USE FOR RANDOM PROCESSING.
OUT-OF-LINE.
    NOTE  THIS IS THE START OF THE OUT-OF-LINE PORTION OF THE
          PROGRAM.
INITIALIZE.
    MOVE ZERO TO SA-ERR-NO.
CALCULATE-ACTUAL-KEY.
    IF ACCOUNT-NUMBER IN SA-TRANS-IN IS NOT NUMERIC
        MOVE 4 TO SA-ERR-NO GO TO FORCE-SEQUENTIAL.
    COMPUTE SA-WORK = SA-AN1 - SA-AN2.
    IF SA-WORK IS NEGATIVE
        COMPUTE SA-WORK = 9999 + SA-WORK ELSE
        COMPUTE SA-WORK = 9999 - SA-WORK.
    IF SA-AN3 = 0
        COMPUTE SA-TRACK-NUMBER = SA-WORK / 10 ELSE
        COMPUTE SA-TRACK-NUMBER = SA-WORK / SA-AN3 * 10.
    NOTE  THIS PARAGRAPH CONVERTED THE SYMBOLIC KEY
          (SA-ACCOUNT-NUMBER) INTO THE ACTUAL KEY
          (SA-TRACK-NUMBER), THE VALUES FOR THE ACTUAL KEY ARE
          BETWEEN 0 AND 999 (1000 TRACKS ARE ASSIGNED TO THE
          FILE).
READ-ACCOUNT-RECORD.
    READ ACCOUNTS INVALID KEY MOVE 1 TO SA-ERR-NO GO TO
        FORCE-SEQUENTIAL.
    MOVE ACC-BALANCE TO SA-OLD-BALANCE.
UPDATE-BALANCE.
    IF DEPOSIT NEXT SENTENCE ELSE GO TO UPDATE-1.
        ADD AMOUNT IN SA-REC TO ACC-BALANCE ON SIZE ERROR
        MOVE 2 TO SA-ERR-NO  GO TO FORCE-SEQUENTIAL.
        GO TO SAVE-RECORD.
  UPDATE-1.
    IF WITHDRAWAL
        SUBTRACT AMOUNT IN SA-REC FROM ACC-BALANCE ELSE
        MOVE 5 TO SA-ERR-NO GO TO FORCE-SEQUENTIAL.
        IF AMOUNT IN SA-REC IS NEGATIVE
            MOVE 6 TO SA-ERR-NO GO TO FORCE-SEQUENTIAL.
SAVE-RECORD.
    MOVE ACC-BALANCE TO SA-NEW-BALANCE.
    REWRITE ACCOUNT-RECORD INVALID KEY MOVE 3 TO SA-ERR-NO.
```

```
FORCE-SEQUENTIAL.
    HOLD.
    NOTE  THE HOLD IS USED TO FORCE SYNCHRONOUS PROCESSING SO
          THAT ANY ERROR MESSAGES PRODUCED WILL APPEAR IN THE
          SAME ORDER AS THE TRANSACTIONS WERE ENTERED AND TO
          PREVENT INTERMIXING LINES OF MESSAGES (I.E. TO INSURE
          THAT WHEN ONE CYCLE DISPLAYS AN ERROR MESSAGE,
          ANOTHER CYCLE DOES NOT GET CONTROL AND ALSO GENERATE A
          MESSAGE).
ERROR-CHECK.
    GO TO E1 E2 E3 E4 E5 E6 DEPENDING ON SA-ERR-NO.
    GO TO END-RANDOM.
  E1.  DISPLAY 'ACCOUNT IS NOT IN ACCOUNTS FILE.'.
       GO TO END-2.
  E2.  DISPLAY 'BALANCE EXCEEDS $9,999,999.99.'.
       GO TO END-1.
  E3.  DISPLAY 'INVALID KEY ERROR TRYING TO REWRITE ACCOUNTS.'.
       DISPLAY 'NEW BALANCE IS ' SA-NEW-BALANCE.
       GO TO END-1.
  E4.  DISPLAY 'ACCOUNT NUMBER IS NOT NUMERIC.'.
       GO TO END-2.
  E5.  DISPLAY 'ILLEGAL TRANSACTION CODE IS SPECIFIED.'.
       GO TO END-2.
  E6.  DISPLAY 'BALANCE OVER-DRAWN.'.
       GO TO END-2.
  END-1.
       DISPLAY 'OLD BALANCE IS ' SA-OLD-BALANCE.
  END-2.
       DISPLAY 'TRANSACTION IS '
        ACCOUNT-NUMBER IN SA-REC '  ' TRANSACTION-CODE IN SA-REC
        '  ' AMOUNT IN SA-REC.
END-RANDOM.
    EXIT.
END DECLARATIVES.

IN-LINE.
    NOTE  THIS IS THE START OF THE IN-LINE PORTION OF THE
          PROGRAM.
START.
    OPEN INPUT TRANSACT I-O ACCOUNTS.
INITIATE-RANDOM-PROCESSING.
    READ TRANSACT AT END GO TO IN-LINE HOLD.
    MOVE CORRESPONDING TRANSACTION-RECORD TO SA-TRANS-IN.
    PROCESS RANDOM-UPDATE.
    GO TO INITIATE-RANDOM-PROCESSING.
IN-LINE-HOLD.
    HOLD RANDOM-UPDATE.
    CLOSE TRANSACT  ACCOUNTS.
    STOP RUN.
```

READER'S COMMENTS

Title: IBM Operating System/360 - COBOL Form: C28-6516-2

Is the material:	Yes	No
Easy to Read?	___	___
Well organized?	___	___
Complete?	___	___
Well illustrated?	___	___
Accurate?	___	___
Suitable for its intended audience?	___	___

How did you use this publication?

___ As an introduction to the subject ___ For additional knowledge

Other _____

fold

Please check the items that describe your position:

___ Customer personnel ___ Operator ___ Sales Representative

___ IBM personnel ___ Programmer ___ Systems Engineer

___ Manager ___ Customer Engineer ___ Trainee

___ Systems Analyst ___ Instructor Other _____

Please check specific criticism(s), give page number(s), and explain below:

___ Clarification on page(s)

___ Addition on page(s)

___ Deletion on page(s)

___ Error on page(s)

Explanation:

CUT ALONG LINE

fold

Name _____

Address _____

FOLD ON TWO LINES, STAPLE AND MAIL

No Postage Necessary if Mailed in U.S.A.

staple staple

fold fold

FIRST CLASS
PERMIT NO. 81

POUGHKEEPSIE, N. Y.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

POSTAGE WILL BE PAID BY

IBM CORPORATION
P. O. BOX 390
POUGHKEEPSIE, N. Y. 12602

ATTN: PROGRAMMING SYSTEMS PUBLICATIONS
DEPT. D58

CUT ALONG LINE

fold fold

staple staple

C28-6516-2

Printed in U.S.A. C28-6516-2

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601