TECHNICAL MANUAL NO 13

PDP-10 COBOL REFERENCE MANUAL

UNIVERSITY OF QUEENSLAND
COMPUTER CENTRE

UNIVERSITY OF QUEENSLAND

Computer Centre

TECHNICAL MANUAL NO 13

PDP-10 COBOL REFERENCE MANUAL

Sarah Barry

This manual is authorized by the Director of the Computer Centre.

MANUAL STATUS

First Edition 1 July 1971

ACKNOWLEDGMENT

COBOL is an industry language and is not the property of any company or group of companies, or of any organization or group of organizations.

No warranty, expressed or implied, is made by any contributor or by the COBOL 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.

Procedures have been established for the maintenance of COBOL. Inquiries concerning the procedures 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 are:

> FLOW-MATIC (trademark of Sperry Rand Corporation), Programming for the Univac (R) I and II, Data Automation Systems copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator Form No. F 28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell.

They have specifically authorized the use of this material, in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications.

FOREWORD

This manual describes COBOL as it has been implemented on the PDP-10. Chapter 1 tells how to get started with COBOL on the PDP-10. Chapter 2 discusses language elements, conventions used in the manual, and the structure of a COBOL program. Chapters 3 through 6 detail the four major divisions of a COBOL program. The COBOL library is described in Chapter 7 and I/O processing is discussed in Chapter 8. Several Appendices have been included in the manual. Appendices A and B contain the COBOL reserved words and the character collating sequence. The ENTER procedure, used to link COBOL programs to programs and subroutines written in other languages, is described in Appendix C. Appendix D details the elements of the COBOL command that invokes the COBOL compiler. Appendices E and F describe programming information of use to COBOL programmers.

It is assumed that the reader has a knowledge of the COBOL language. This manual is intended primarily for reference and is not a tutorial guide for beginning COBOL programmers. Those wishing to learn the COBOL language are referred to the following books.

> Davis, G.B. and Litecky, C.R., *Elementary COBOL Programming - A Step by Step Approach*, McGraw-Hill, Inc., 1971.
>
> Stern, Nancy B. and Stern, Robert A., *COBOL Programming*, John Wiley and Sons, Inc., 1970.
>
> McCracken, Daniel D. and Garbassi, Umberto, *A Guide to COBOL Programming*, Second Edition, New York, John Wiley and Sons, Inc., 1970.

The COBOL programmer should be familiar with the operating system commands and the editing language of the PDP-10. The commands are discussed in the System User's Guide, MNT-8; the Editor is described in MNT-6.

This manual has been adapted from the Digital Equipment Corporation manual. While some of the features are not available on the University of Queensland computer system they have been included in this manual for the sake of conformity with the standard COBOL requirements.

The use of console switches is not permitted either through Batch or Teletypes. In a timesharing environment several users might require the same switches to be set differently at the same period of time. This is most impractical and unnecessary.

The only peripherals available to users are the disk and the Teletype to terminal users; and the disk and card reader to batch users. The line printer can be used by both. Batch operates via a phenomenon known as the *pseudo-Teletype*. Users running programs through Batch must assign their card input files to TTY and their line printer output files to TTY. Care must be taken that two files assigned to TTY are not both open at the same time. This problem can be overcome by assigning the printer output file to the disk and then listing it at the termination of the program by the LIST command.

Chapter 7 discusses the COPY verb and the COBOL library. At present, users cannot create their own libraries nor maintain them. Chapter 7 has been left in for completeness only. Later revisions to the COBOL manual will give full details of library creation and maintenance.

CONTENTS

CHAPTER 5 THE DATA DIVISION

CHAPTER 6 THE PROCEDURE DIVISION

CHAPTER 7 THE COBOL LIBRARY

CHAPTER 8 STANDARD I-O PROCESSING

CHAPTER 8 contd

ILLUSTRATIONS

TABLES

CHAPTER 1

GETTING STARTED WITH PDP-10 COBOL

COBOL (Common-Business-Oriented Language) is an industry-wide data processing language designed for business applications, such as payroll, inventory control, and accounts-receivable. In COBOL, the programmer can describe his data and the processing of it in simple, English-like statements. COBOL programs are written in terms that are easily recognized by the business user; thus, little programmer training is required, the programs are virtually self-documenting, and programming of desired applications is accomplished quickly and easily.

1.1 PDP-10 COBOL

The COBOL implemented for the PDP-10 conforms to American National Standards Institute (ANSI) specifications as described in the document USA Standard COBOL, X3.23-1968.

PDP-10 COBOL operates within the PDP-10 operating system, thereby offering the COBOL user the business processing capability of COBOL, in addition to the many features of the PDP-10. Some of these features are:

a. Batch and timesharing modes of operation,

b. On-line editing and debugging,

c. Sharable compiler and object code.

1.1.1 Modes of Operation

The PDP-10 operating system supports both interactive and batch modes of operation. The COBOL programmer can use either or both of these operating environments to develop his applications. In a timesharing system, a program can be written at an interactive terminal, edited and debugged while the usr is on-line, and then run immediately. The turn-around time normally associated with preparing an error-free program can be substantially reduced, because the programmer receives the results of his program immediately and can determine, on the spot, whether or not his program is running properly.

Programs that need no interaction with the user, that require large amounts of data, or that are very long can be entered into the batch system through a noninteractive device such as the card reader. Through commands in his batch job, the programmer can specify the processing that he requires, the constraints that must be fulfilled (e.g., deadlines, priorities, and time limits), and the devices that are necessary. The normal tasks required of the interactive terminal user (e.g., logging in and out) are performed by the batch system.

Whether the user needs fast interaction with the system or rapid throughput, the PDP-10 operating system can offer him both.

1.1.2 On-Line Editing and Debugging

To develop error-free programs, the COBOL user can take advantage of the system programs Editor and DDT. Editor is the system editing program, and DDT (Dynamic Debugging Technique) is the system program used for debugging programs during execution.

Editor is a simple, easily-learned program for editing disk files. It performs editing on lines in these files. Within a line, the user can add, delete, and change characters. New lines can be added to the file and unwanted lines can be deleted by means of simple commands. The interactive user, when creating programs at his terminal, must use the editor to place his programs into files on disk.

DDT is a system program that allows the user to perform checkout and debugging of his programs. By typing commands to DDT, the user can set breakpoints in his program and examine and modify the contents of any location. At the user's option, insertion and deletion of code can be performed in source language or in various numeric or text modes. DDT is described in detail in the Utility Programs manual, section 3 of MNT-12.

1.1.3 Sharable Code

The COBOL compiler, like most of the system software, is divided into high and low segments so that most of its code can be shared by many users. The high segment of the compiler is reentrant; that is, it contains code that all COBOL users share. The low segment, containing tables, data names, procedure names, and the like, is unique to each user. This reentrant capability means that only one copy of the compiler must be resident in core at any one time to serve all COBOL users, thus minimizing the amount of core used by COBOL compilations.

The user can, if he desires, write his COBOL programs so that they are also sharable. This feature is advantageous if a great many people need to use the same program simultaneously.

1.1.4 Peripheral Devices

The PDP-10 COBOL user has available to him a choice of peripheral devices, whether he is operating in interactive or batch mode. Programs and data can be entered from interactive terminals, paper-tape reader, card reader, or disk. Program listings and program output can be printed on the line printer and on the user's interactive terminal. They can be stored on paper tape, or disk.

The PDP-10 System has been designed to provide maximum speed, efficiency, and ease of operation for a large number of simultaneous batch and timesharing users. Because COBOL has been fully integrated into this system, the COBOL programmer can take advantage of the full range of capabilities provided by the PDP-10, in addition to the power of the high-level, ANSI-standard, PDP-10 COBOL.

1.2 USING COBOL ON THE PDP-10

Two sample COBOL runs are shown below. The first demonstrates operations in timesharing mode; the second shows batch mode.

1.2.1 COBOL Programming in Timesharing Mode

A sample COBOL program is illustrated and described below, along with all the steps necessary to create, compile and execute it. This example shows an interactive session, using the timesharing mode of operation. The source program format is the non-standard format, not the conventional COBOL format. Both formats are acceptable to the COBOL compiler; they are described in Chapter 2. The sample run is shown in its entirety first. Each part is then described in detail. Input from the user is underlined in all of the examples.

```
.LOGIN<cr>
JOB 11     MON 3/17 MAS
PROJECT    #46<cr>
COST LIM:  1Ø.ØØ<cr>
PASSWORD:  password<cr>

IDENT. TTT14    ↑O

.CREATE SALES1/CBL<cr>
INPUT:
IDENTIFICATION DIVISION.<cr>
.
.
.
```

```
<tab   >STOP RUN.<cr>
<cr>
*FILE<cr>

EXIT
↑C

.COBOL(LIST, NONSTD) SALES1/CBL, LST=SALES1/LST<cr>
COBOL:  SALES1

EXIT
↑C

.RUN<cr>
LOADING

LOADER 2K CORE
EXECUTION

EXIT
↑C

.LIST SALES1/LST, SLSMRS<cr>
-LISTING THESE FILES-

SALES1/LST
SLSMRS

EXIT
↑C
```

.

The first step is logging in to the timesharing system. The monitor types a period to indicate that it is ready to receive a command. The user types the LOGIN command with his project number and cost limit. The system returns a job number, the version of the monitor, the system it is running on, and then requests the user's password. The password is not echoed to the terminal to preserve the security of the user's project number. After the correct password is typed, the system types the date, time, and any pertinent messages.

```
.LOGIN<cr>
JOB 11    MON 3/17 MAS
PROJECT #46<cr>
COST LIM:1Ø.ØØ<cr>
PASSWORD:password<cr>

IDENT. TTY14  ↑O
```

After logging in, the user creates his COBOL program through the Editor. The user issued the CREATE monitor command to cause the Editor to open a disk file with the specified name and extension. When the Editor returns the message 'INPUT:', the user can enter his program by typing the text of his program.

```
.CREATE SALES1/CBL<cr>
INPUT:
IDENTIFICATION DIVISION.<cr>
.
.
.
```

To end the insert (his program), the user must press an extra return which transfers control to edit mode. To close the file and return to the monitor, the user types FILE.

```
<cr>
*FILE<cr>

EXIT
↑C
```

.

The sample program uses a transaction file SLSTRS to update a master file SLSMRF. The resulting master file produced is called SLSMRS. All three files are maintained on disk and on completion of the program the new master file is listed on the line printer.

To compile, load, and execute his COBOL program, the user issues the COBOL and RUN monitor commands. To signal that the program has completed execution, the system types EXIT.

```
.COBOL(LIST, NONSTD) SALES1/CBL, LST=SALES1/LST<cr>
COBOL:  SALES1

EXIT
↑C

.RUN<cr>
LOADING

LOADER 2K CORE
EXECUTION

EXIT
↑C
```

.

The system will automatically enter the compilation listing of the program and the son master file into the line printer queue when the user issues the LIST command.

```
.LIST SALES1/LST SLSMRS<cr>
-LISTING THESE FILES-

SALES1/LST
SLSMRS

EXIT
↑C

.
```

The compilation listing is shown below together with a portion of the output master file. Note that although the original file did not have sequence numbers the compiler has supplied them automatically.

P R O G R A M S A L E S 1

```
ØØØ1      IDENTIFICATION DIVISION.
ØØØ2      PROGRAM-ID.    SALES1.
ØØØ3      REMARKS.       THIS PROGRAM UPDATES THE SALES MASTER FILE.
ØØØ4      ENVIRONMENT DIVISION.
ØØØ5      INPUT-OUTPUT SECTION.
ØØØ6      FILE-CONTROL.
ØØØ7              SELECT OLD-MASTER     ASSIGN TO DSK.
ØØØ8              SELECT NEW-MASTER     ASSIGN TO DSK.
ØØØ9              SELECT TRANSACTION    ASSIGN TO DSK.
ØØ1Ø      DATA DIVISION.
ØØ11      FILE SECTION.
ØØ12      FD      OLD-MASTER
ØØ13                    LABEL RECORDS ARE STANDARD
ØØ14                    VALUE OF IDENTIFICATION IS "SLSMRF   "
ØØ15                    DATA RECORD IS OLD-MAST.
ØØ16      Ø1      OLD-MAST              DISPLAY-7.
ØØ17              Ø2 PRODUCT-NUMBER     PICTURE X(5).
ØØ18              Ø2 UNIT-PRICE         PICTURE 999V99.
ØØ19              Ø2 YTD-SALES          PICTURE 9(5)V99.
ØØ2Ø      FD      NEW-MASTER
ØØ21                    LABEL RECORDS ARE STANDARD
ØØ22                    VALUE OF IDENTIFICATION IS "SLSMRS   "
ØØ23                    DATA RECORD IS NEW-MAST.
ØØ24      Ø1      NEW-MAST              DISPLAY-7.
ØØ25              Ø2 PRODUCT-NUMBER     PICTURE X(5).
ØØ26              Ø2 UNIT-PRICE         PICTURE 999V99.
ØØ27              Ø2 YTD-SALES          PICTURE 9(5)V99.
```

```
ØØ28      FD      TRANSACTION
ØØ29                    LABEL RECORDS ARE STANDARD
ØØ3Ø                    VALUE OF IDENTIFICATION IS "SLSTRS   "
ØØ31                    DATA RECORD IS TRANS.
ØØ32      Ø1      TRANS                  DISPLAY-7.
ØØ33              Ø2 PRODUCT-NUMBER      PICTURE X(5).
ØØ34              Ø2 QUANTITY            PICTURE 999.
ØØ35              Ø2 SALESMAN            PICTURE XX.
ØØ36              Ø2 DISTRICT            PICTURE X.
ØØ37      WORKING-STORAGE SECTION.
ØØ38      77      TOTAL-PRICE            PICTURE 9(5)V99 VALUE ZEROS.
ØØ39      77      PRODUCT-TOTAL          PICTURE 9(5)V99 VALUE ZEROS.
ØØ4Ø      77      SAVE-TRANS-PROD-NO     PICTURE X(5).
ØØ41      PROCEDURE DIVISION.
ØØ42      HOUSEKEEPING.
ØØ43              OPEN INPUT TRANSACTION, OLD-MASTER
ØØ44                     OUTPUT NEW-MASTER.
ØØ45              MOVE SPACES TO NEW-MAST.
ØØ46              READ TRANSACTION RECORD, AT END GO TO LAST-TRANSACTION.
ØØ47              MOVE PRODUCT-NUMBER IN TRANSACTION TO SAVE-TRANS-PROD-NO.
ØØ48      READ-OLD-MASTER.
ØØ49              READ OLD-MASTER RECORD, AT END GO TO WRAPUP.
ØØ5Ø      TESTING-1.
ØØ51              IF SAVE-TRANS-PROD-NO IS EQUAL TO PRODUCT-NUMBER
ØØ52                     IN OLD-MAST GO TO PROCESS.
ØØ53      TESTING-2.
ØØ54              IF PRODUCT-NUMBER IN OLD-MAST IS LESS THAN
ØØ55                     SAVE-TRANS-PROD-NO WRITE NEW-MAST FROM OLD-MAST
ØØ56                     GO TO READ-OLD-MASTER, ELSE STOP
ØØ57                     "FILE SEQUENCE ERROR OR INVALID TRANS PRODUCT NUMBER".
ØØ58      PROCESS.
ØØ59              MULTIPLY UNIT-PRICE IN OLD-MAST BY QUANTITY
ØØ6Ø                     GIVING TOTAL-PRICE.
ØØ61              ADD TOTAL-PRICE TO PRODUCT-TOTAL.
ØØ62              READ TRANSACTION RECORD, AT END GO TO LAST-TRANSACTION.
ØØ63              MOVE PRODUCT-NUMBER IN TRANSACTION TO SAVE-TRANS-PROD-NO.
ØØ64              IF SAVE-TRANS-PROD-NO IS EQUAL TO PRODUCT-NUMBER
ØØ65                     IN OLD-MAST GO TO PROCESS.
ØØ66      DIFFERENT-PRODUCT.
ØØ67              ADD PRODUCT-TOTAL TO YTD-SALES IN OLD-MAST.
ØØ68              WRITE NEW-MAST FROM OLD-MAST.
ØØ69              MOVE ZEROS TO PRODUCT-TOTAL.
ØØ7Ø              GO TO READ-OLD-MASTER.
ØØ71      LAST-TRANSACTION.
ØØ72              MOVE HIGH-VALUES TO SAVE-TRANS-PROD-NO.
ØØ73              GO TO DIFFERENT-PRODUCT.
ØØ74      WRAPUP.
ØØ75              CLOSE TRANSACTION, OLD-MASTER, NEW-MASTER.
ØØ76              STOP RUN.

NO ERRORS DETECTED
```

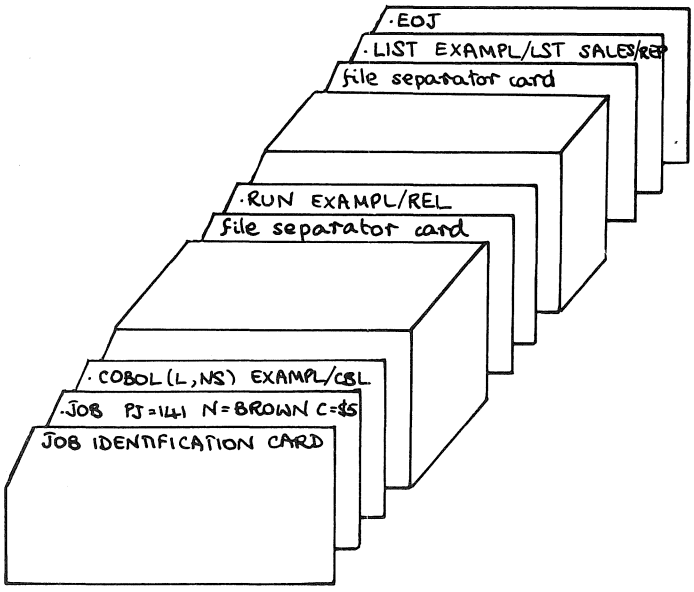
```
                S L S M R S

A2315ØØ1ØØØØØØØØØ
A329ØØØØ95ØØ1Ø45Ø
B9266Ø3575ØØØ715Ø
B9754 . . .
.
.
.
```

At the end of this task, the user can log off the system or he can continue to perform other tasks at his terminal.

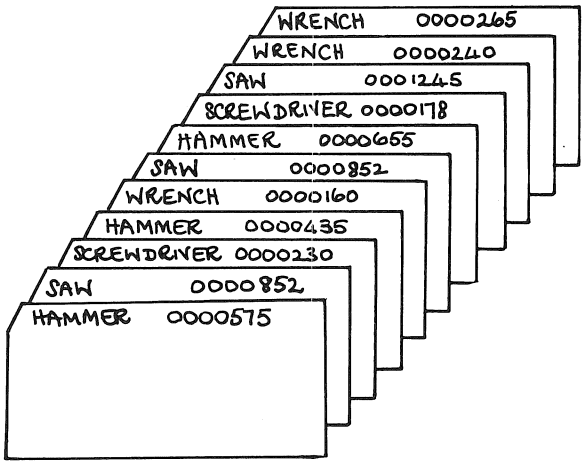
1.2.2 COBOL Programming in Batch Mode

The sample program described in this section shows the steps that are necessary to run a program using the Multiprogramming Batch system. The input card deck, which contains the program, data, and the batch control commands, is shown first. Each part of the sample deck is then described in detail. The output from the program, and a listing of the program are shown below.

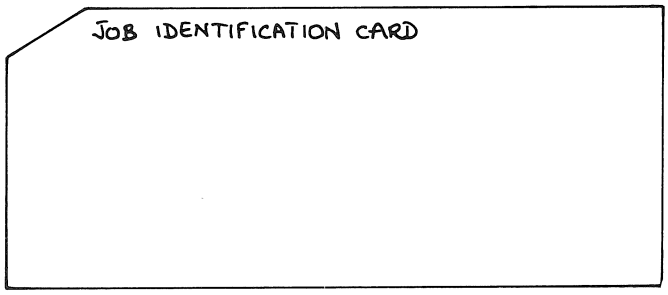


The programmer writes his program and punches it on cards in ASCII code to be input to the Batch system. The sample program is coded in non-standard format as in the previous example. This program reads a list of items sold, sorts them into ascending alphabetic order, and prints a report containing the items grouped according to type, the price of each item, and the subtotal for each group of items. The program requests display of the number of items sold and the gross income from the sales.

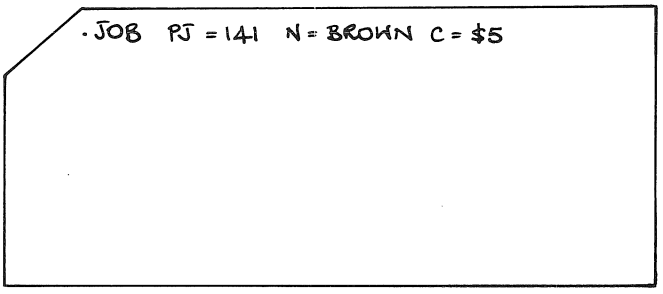
The data for the program, which is a group of items sold, is also placed on cards. This data is shown below.



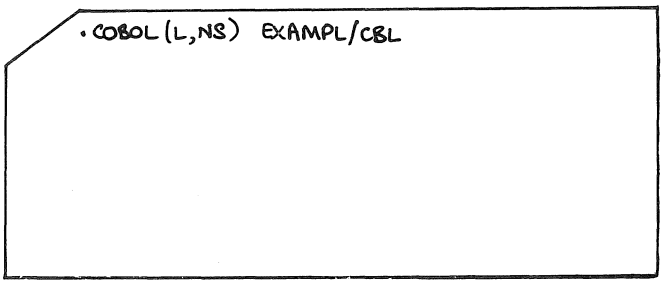
The programmer sets up his job as an input deck containing his program, data, and Batch control commands on cards. The following Batch commands are used for this job.



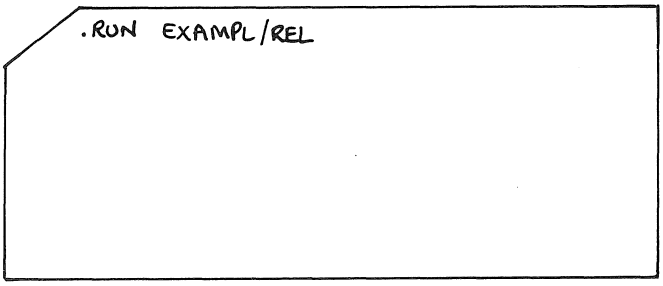
The Job Identification card contains the job's sequence number. This card is compulsory.



The .JOB command contains the user's name and project number, and the cost limit for the job.



The .COBOL command contains the name of the COBOL program. This command is followed immediately by the deck containing the program. When the job is run, the program is compiled, and a listing is produced. The listing is printed with the output of the job.



The .RUN command is immediately followed by the deck containing the data. The .RUN command causes the program (or programs) that precedes it to be executed when the job is run by the batch controller.

The .LIST command will transfer the listing of the program and the output of the program from the user's disk area for later printing by the line printer symbiont program.

The end of the job is signalled by the .EOJ card.

The user inputs his job deck through the card reader and then simply waits for his output. The Multiprogramming Batch system reads the cards, runs the job, and spools the output to the line printer. The output is in the form of a printed listing that begins with a header page and ends with a trailer page. Both the header and trailer contain the user's name, project number, date, time, system name, and monitor version. Between the header and trailer are the compilation listing, the loader map, and the output from program. With the exception of the header and trailer pages and the loader map, the listing is shown. Note that the compiler assigns sequence numbers to the compilation listing even though the user did not place sequence numbers on his program.

P R O G R A M E X A M P L

```
ØØØ1      IDENTIFICATION DIVISION.
ØØØ2      PROGRAM-ID. EXAMPL.
ØØØ3      REMARKS, READS A LIST OF SALES AND PRINTS
ØØØ4               A REPORT WITH SUBTOTALS FOR ITEM TYPES.
ØØØ5      ENVIRONMENT DIVISION.
ØØØ6      CONFIGURATION SECTION.
ØØØ7      SOURCE-COMPUTER. PDP-1Ø.
ØØØ8      INPUT-OUTPUT SECTION.
ØØØ9      FILE-CONTROL.
ØØ1Ø            SELECT SALES,   ASSIGN TO TTY.
ØØ11            SELECT TEMP,    ASSIGN TO DSK, DSK, DSK.
ØØ12            SELECT REPT,    ASSIGN TO TTY.
ØØ13      DATA DIVISION.
ØØ14      FILE SECTION.
ØØ15      FD    SALES, VALUE OF IDENTIFICATION IS "SALES CDS".
ØØ16      01    SALES-CARD, DISPLAY-7.
ØØ17            Ø2 ITEM,      PIC X(2Ø).
ØØ18            Ø2 PRICE,     PIC 9(5)V99.
ØØ19      SD    TEMP.
ØØ2Ø      Ø1    SORT-REC.
ØØ21            Ø2 ITEM,      PIC X(2Ø).
ØØ22            Ø2 FILLER,    PIC 9(7).
ØØ23      FD    REPT, VALUE OF IDENTIFICATION IS "SALES REP".
ØØ24      Ø1    REPT-LINE.
ØØ25            Ø2 R-ITEM,    PIC X(2Ø).
ØØ26            Ø2 FILLER,    PIC X(5).
ØØ27            Ø2 R-PRICE,   PIC Z(8).99.
ØØ28      Ø1    HEADER, PIC X(36).
ØØ29      WORKING-STORAGE SECTION.
ØØ3Ø      Ø1    REC-STORE.
ØØ31            Ø2 REC-ITM,     PIC X(2Ø),
ØØ32            Ø2 REC-PR,      PIC 9(5)V99.
ØØ33      77    NUM,            PIC 9(6).
ØØ34      77    GROSS-TMP,      PIC 9(8)V99.
```

```
ØØ35       77     GROSS,          PIC Z(8).99
ØØ36       77     SUBTOT,         PIC 9(8)V99.
ØØ37       77     LAST-ITEM,      PIC X(2Ø).
ØØ38       PROCEDURE DIVISION.
ØØ39       MAIN SECTION
ØØ4O       BEGIN.
ØØ41              SORT TEMP ON ASCENDING KEY ITEM OF SORT-REC,
ØØ42                   USING SALES,
ØØ43                   OUTPUT PROCEDURE MAKE-REPORT.
ØØ44              DISPLAY "NUMBER OF ITEMS SOLD    ", NUM.
ØØ45              MOVE GROSS-TMP TO GROSS.
ØØ46              DISPLAY "GROSS INCOME            ", GROSS.
ØØ47              STOP RUN.
ØØ48       *      OUTPUT PROCEDURE FOR SORT
ØØ49       MAKE-REPORT SECTION.
ØØ5Ø       INIT.
ØØ51              OPEN OUTPUT REPT.
ØØ52              MOVE ZERO TO NUM, SUBTOT, GROSS-TMP.
ØØ53              MOVE "SALES REPORT" TO HEADER.
ØØ54              WRITE HEADER BEFORE ADVANCING 2 LINES.
ØØ55       LOOP.
ØØ56              RETURN TEMP INTO REC-STORE, AT END GO TO FINIS.
ØØ57              IF GROSS-TMP = Ø, GO TO LOOP-2.
ØØ58              IF REC-ITM NOT = LAST-ITEM
ØØ59                   MOVE SPACES TO R-ITEM,
ØØ6Ø                   MOVE SUBTOT TO R-PRICE,
ØØ61                   MOVE ZERO TO SUBTOT,
ØØ62                   WRITE REPT-LINE BEFORE ADVANCING 2 LINES.
ØØ63       LOOP-2.
ØØ64              SET NUM UP BY 1.
ØØ65              ADD REC-PR TO GROSS-TMP
ØØ66              ADD REC-PR TO SUBTOT.
ØØ67              MOVE REC-ITM TO R-ITEM.
ØØ68              MOVE REC-PR TO R-PRICE.
ØØ69              WRITE REPT-LINE.
ØØ7Ø              MOVE REC-ITM TO LAST-ITEM.
ØØ71              GO TO LOOP.
ØØ72       FINIS.
ØØ73              MOVE SPACES TO R-ITEM.
ØØ74              MOVE SUBTOT TO R-PRICE.
ØØ75              WRITE REPT-LINE BEFORE ADVANCING 2 LINES.
ØØ76                              MOVE "TOTAL" TO R-ITEM.
ØØ77                                   MOVE GROSS-TMP TO R-PRICE.
ØØ78              WRITE REPT-LINE.
ØØ79              CLOSE REPT.

NO ERRORS DETECTED

SALES REPORT

HAMMER                          5.75
HAMMER                          4.35
HAMMER                          6.55
                               16.65
```

SAW	8.52
SAW	12.45
SAW	8.52
	29.49
SCREW DRIVER	2.30
SCREW DRIVER	1.78
	4.08
WRENCH	1.60
WRENCH	2.65
WRENCH	2.40
	6.65
TOTAL	56.87

Chapter 2
Introduction to COBOL Language

The conventions, special terms, language elements, and formats acceptable to COBOL are described below to aid the user in writing COBOL programs. The source language statements are discussed in subsequent chapters.

2.1 SYMBOLS AND TERMS

The symbols and terms used in the following chapters of this manual are either necessary to describe the language or are commonly-used COBOL terms.

2.1.1 Symbols

The symbology used in this manual to illustrate the various COBOL statement formats is essentially the same as that used in other COBOL language manuals and is based on the CODASYL COBOL reference document.

Symbology	Meaning
Lower-case characters	Represent information that must be supplied by the programmer, such as values, names, and other parameters.
Upper-case characters, underscored	Key words in the COBOL lexicon that must be used when the formats of which they are a part are used.
Upper-case characters, not underscored	Other words in the COBOL lexicon that serve only to make the COBOL statement more readable. Their use is optional and has no effect on the meaning of the formats of which they are a part.
Braces	Indicate that a choice must be made from the two or more lines enclosed.

(continued on next page)

Symbology	Meaning
Brackets	Indicate an optional feature. The contents of the brackets are used according to the rules above if the feature is desired.
Ellipsis ...	Indicates that the information contained within the preceding pair of braces or brackets can be repeated at the programmer's option.

2.1.2 COBOL Terms

The terms block, record, and item have special meanings when used in a COBOL program.

Term	Meaning
Block	Signifies a physical grouping of records. This term commonly refers to a physical block of records on some storage medium.
Record	Signifies a logical unit of information. In relation to a data file, a record is the largest unit of logical information that can be accessed and processed at a time. Records can be subdivided into fields or items.
Item	Signifies a logical field or group of fields within a record. A group item is one that is further broken down into subitems (e.g., a group item called TAX might be broken down into subitems called FED-TAX and STATE-TAX). Subitems can be further broken down into other subitems. An item that has no subitems is called an elementary item.

2.2 ELEMENTS OF COBOL LANGUAGE

2.2.1 Program Structure

A COBOL program consists of four divisions, all of which must be present. Within each division are the program statements; some are required, others are optional.

Division	Meaning
IDENTIFICATION DIVISION	Identifies the source program.
ENVIRONMENT DIVISION	Describes the computer on which the source program is to be compiled, the computer on which the object program is to run, and certain relationships between program elements and hardware devices.
DATA DIVISION	Describes the data to be processed by the object program.
PROCEDURE DIVISION	Describes the actions to be performed on the data.

2.2.2 Character Set

Within a source program statement, the ASCII characters are valid except:

a. null, delete, and carriage return (which are ignored);

b. line feed, vertical tab, form feed, and the printer control characters (20_8 through 24_8), which mark the end of a source line;

c. Control-Z, which marks the end-of-file.

The lower case ASCII characters are translated to upper case characters except when they appear in non-numeric literals.

Of this character set, 37 characters (the digits 0 through 9, the 26 letters of the alphabet, and the hyphen) can be used by the programmer to form COBOL words, such as data-names, procedure-names, and identifiers.

Punctuation characters include:

Δ	(space)	"	(quotation mark)
,	(comma)	(	(left parenthesis)
;	(semicolon)	)	(right parenthesis)
.	(period)	→|	(horizontal tab)

Special editing characters include:

+	(plus sign)	*	(check protection symbol)
-	(minus sign)	Z	(zero suppression)
$	(dollar sign)	B	(blank insertion)
,	(comma)	0	(zero insertion)
.	(decimal point)	CR	(credit)
		DB	(debit)

Special characters used in arithmetic expressions include:

+	(addition)	/	(division)
-	(subtraction)	**	(exponentiation)
*	(multiplication)	↑	(exponentiation)

Special characters used in conditional (IF) statements include:

=	(equal)	>	(greater than)	<	(less than)

2.2.3 Words

A COBOL word is composed of not more than 30 characters chosen from the 37 characters given in the previous section. A word is terminated by a space, period, right parenthesis, comma, semicolon, or horizontal tab. If the terminator is not a space or horizontal tab, at least one space or tab must follow the terminator.

Words used in writing COBOL source programs are of two types: COBOL reserved words and user-created words.

2.2.3.1 COBOL Reserved Words - COBOL reserved words are those words that constitute the COBOL lexicon and have a special meaning to the compiler (e.g., DIVISION, PROCEDURE, ADD); these words are listed in Appendix A. They include all the COBOL division, section, and paragraph names, descriptive clauses, procedure verbs, certain prepositions, figurative constants, and special registers. Reserved words must be spelled and used exactly as shown in the formats given in this manual.

2.2.3.1.1 Figurative Constants - Figurative constants are reserved words that specify certain fixed values. When these reserved words are to be used as figurative constants, they must not be enclosed in quotation marks; otherwise they will be treated by the compiler as nonnumeric literals.

The figurative constants are given below. Except for one case (the ALL constant), singular and plural forms are given; these forms are equivalent and can be used interchangeably.

Figurative Constant	Use
ZERO ZEROS ZEROES	Represents the value 0 or one or more of the character 0 depending on context.
SPACE SPACES	Represents one or more blanks or spaces.
HIGH-VALUE HIGH-VALUES	For DISPLAY-6 and DISPLAY-7 items this represents the highest value in the collating sequence. For COMP and COMP-1 items, this represents the largest number that can be placed in the machine word(s) containing the item.
LOW-VALUE LOW-VALUES	For DISPLAY-6 and DISPLAY-7 items, this represents the lowest value in the collating sequence. For COMP and COMP-1 items, this represents the smallest number (most negative) that can be placed in the machine word(s) containing the item.

(continued on next page)

Figurative Constant	Use
QUOTE QUOTES	Represents one or more quotation marks ("). It can be used anywhere that the quotation mark character (") is valid, except to delimit nonnumeric literals (see Nonnumeric Literals). QUOTE(S) is frequently used where an actual quotation mark character would erroneously appear to delimit a nonnumeric literal. For example, if the user wanted his program to type out the exact character string MOUNT TAPE LABELLED "MASTER" ON DRIVE 3 he could use the procedure statement DISPLAY "MOUNT TAPE LABELLED " QUOTE "MASTER" QUOTE " ON DRIVE 3".
ALL any-literal	Represents repetitions of the string of characters that constitute either a nonnumeric literal or a figurative constant (other than ALL any-literal). If a figurative constant is used, the ALL is redundant; thus, ZEROS and ALL ZEROS are equivalent.

Figurative constants generate a string of characters whose length is determined, based on context, by the compiler. For example, if TOTAL-AMOUNT is a five-character field, the procedure statement MOVE ALL ZEROS TO TOTAL-AMOUNT moves a string of five zeros to the field TOTAL-AMOUNT; MOVE ALL "AB" TO TOTAL-AMOUNT moves "ABABA" to TOTAL-AMOUNT. If the length cannot be determined by context, a single character (or a single-character sequence, in the case of ALL) is generated. For example, the procedure statement DISPLAY ALL QUOTES will result in the output of a single quotation mark (") to the user's terminal.

Examples of Use of Figurative Constants:

DATA DIVISION Usage:	02 AMOUNT PICTURE IS 9999.99 VALUE IS ZERO. 04 MESSAGE PICTURE IS A(10) VALUE IS SPACES.
PROCEDURE DIVISION Usage:	MOVE ZEROS TO AMOUNT. (Puts the value of zero in the AMOUNT field) MOVE SPACES TO MESSAGE. (Puts spaces in the MESSAGE field) IF TOTAL IS EQUAL TO ZERO.... EXAMINE FLD-A TALLYING LEADING ZEROS.

2.2.3.1.2 Special Registers - In addition to figurative constants, COBOL recognizes two other special reserved-word constants: TALLY and TODAY.

TALLY is the name of a fixed five-digit signed computational field. It is used primarily to hold information produced by the EXAMINE verb. However, the programmer can utilize TALLY in any situation where a signed numeric field is valid (e.g., temporary storage of any integer value of five or fewer digits).

TODAY is a 12-character alphanumeric display field that contains the current date and time. Its format is:

yymmddhhmmss

where	yy is the year (last two digits)	hh is the hour
	mm is the month	mm is the minute
	dd is the day	ss is the second

2.2.3.2 User-Created Words - User-created words are labels for the various parts of the user's data (files, records, and fields) and the user's procedures (sections and paragraphs). They can contain only the symbols 0 through 9, A through Z, and the hyphen. With the exception of procedure names, they cannot be all digits. A user-created word can neither begin nor end with a hyphen.

User-created words can be further subdivided into several categories. To understand the remainder of this manual, the user should be familiar with the following types of words.

data-name	The user-created name assigned to an item (field) within a record.
file-name	The user-created name assigned to a data file.
record-name	The user-created name assigned to a data record within a file.
procedure-name	The user-created name assigned to a paragraph or section in the PROCEDURE DIVISION. When assigned to a section, it is referred to as a section-name; and when assigned to a paragraph, it is referred to as a paragraph-name.
identifier	A user-created name used in PROCEDURE DIVISION statement formats to indicate a data-name followed, as required, by the syntactically correct combination of qualifiers, and/or subscripts, and/or indexes necessary to make reference to a unique item of data.
mnemonic-name	A user-created name assigned to a hardware device.
condition-name	A user-created name assigned to a value or range of values of the associated data item. Condition-names can also be assigned to console switch settings.
index-name	A user-created name defined in the INDEXED BY clause (see OCCURS in Chapter 5).
index data-name	A user-created name defined with USAGE INDEX.

2.2.4 Literals

A literal is a string of characters, the value of which is identical to the characters that compose the literal. Literals are of two types: numeric and nonnumeric.

2.2.4.1 Numeric Literals - A numeric literal is a string of 1 to 18 numeric characters (0 through 9). It cannot contain any alphabetic characters. It can be preceded by a plus sign (+) or a minus sign (-); if no sign is used, the literal is assumed to be positive. A decimal point can appear anywhere in the literal except to the left of the sign or as the rightmost character. If no decimal point is used, the literal is assumed to be an integer. A numeric literal is considered to be of the numeric class; that is, it can be used legitimately as a value in arithmetic expressions.

Examples of Numeric Literals:

```
123          -123          +123          1.23456          .123456789
-.123456789          1234567890.12345678          -1234567890.12345678
```

2.2.4.2 Nonnumeric Literals - Nonnumeric literals are character strings containing from 1 to 120 characters enclosed in quotation marks. The value of the literal is equal to the characters, including any spaces, enclosed by the quotation marks. Any ASCII character except the quotation mark, null, delete, carriage return, and printer control can appear within a literal.

All nonnumeric literals are considered to be in the alphanumeric category; they cannot be used as values in arithmetic operations, and numeric editing cannot be performed on them. If a literal conforms to the rules for formation of a numeric literal, but is enclosed in quotation marks, it is considered to be a nonnumeric literal. That is, "120.45" is not equivalent to 120.45.

Examples of Nonnumeric Literals:

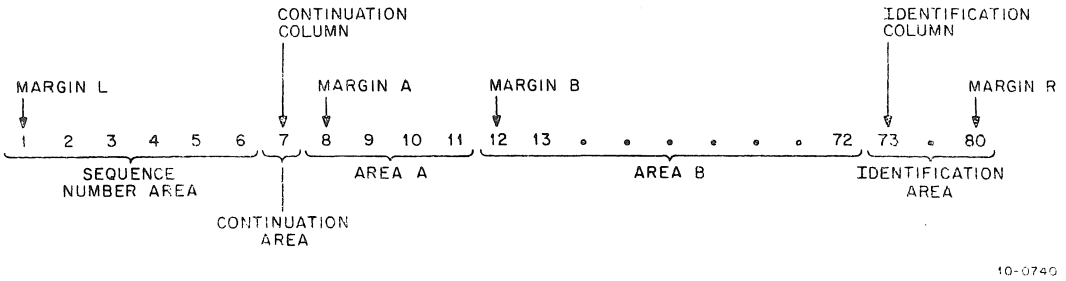
```
"A"    "THIS ACCOUNT HAS A CREDIT BALANCE"    "  RETURN  "
"-125.50"      "DEDUCT 10% IF PAID BEFORE JAN. 31ST"
```

2.3 SOURCE PROGRAM FORMAT

Two source program formats are acceptable to PDP-10 COBOL: standard format and non-standard format. The compiler assumes that the source program is written in standard format unless the NONSTD option is used in the command string (refer to Appendix D).

2.3.1 Standard Format

The standard format is used when the programmer wishes his source programs to be compatible with other COBOL compilers. It is the format that is normally used when input is from the card reader. A line of standard format is shown below; the numbers refer to card columns, although this format can be used with any input medium.



Margin L designates the leftmost (first) character position of a line.

The continuation column designates the seventh character position relative to the left margin.

Margin A designates the eighth character position relative to the left margin.

Margin B designates the twelfth character position relative to the left margin.

The identification column designates the seventy-third character position relative to the left margin.

Margin R designates the rightmost (eightieth) character position of a line.

The sequence number area is a six-character field beginning at margin L that normally contains a sequence number. The compiler ignores this field.

The continuation area occupies one character position in the continuation column. The continuation area is used whenever it is necessary to split a word, a numeric literal, or a nonnumeric literal between the end of one line and the beginning of the next, or when a comment line is to be inserted. The following rules apply to comment or continuation lines.

a. The programmer can insert a comment line in a program by placing an asterisk (*) in the continuation area.

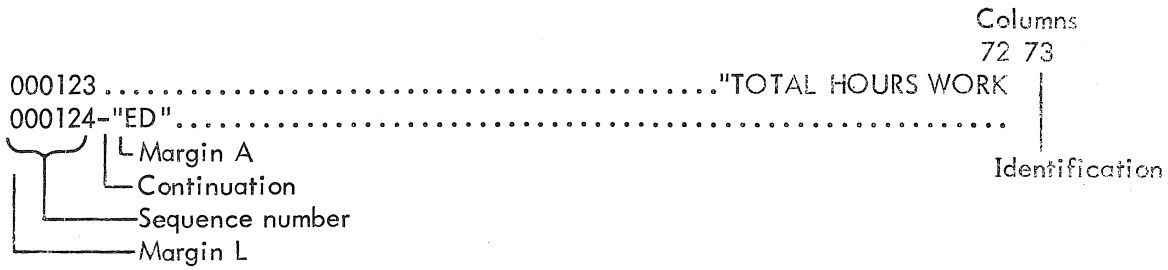
b. To continue a line without splitting a word or literal, the programmer must begin the first continuing word on the second line at, or after, margin A. The continuation area is left blank. As many spaces as desired can follow the last word on the first line, or the word can continue up to the identification column (column 73).

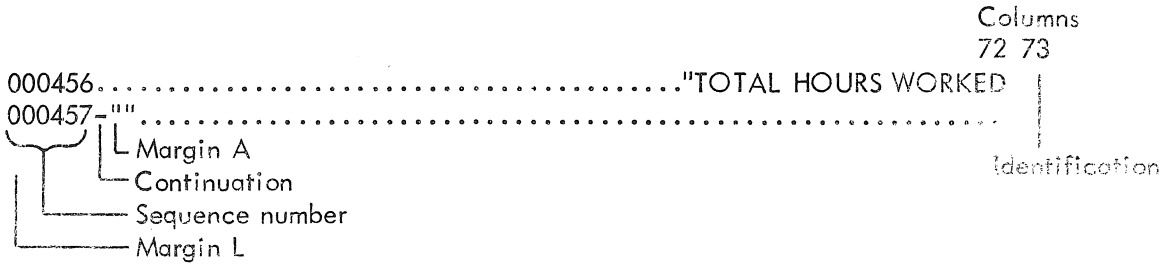
(continued on next page)

c. To split a word or numeric literal from one line to the next, the programmer places a hyphen in the continuation area of the second line and begins the first continuing character of the word or literal at, or after, margin A. As many spaces as desired can occur after the last character of the word or numeric literal on the first line, or the last character can occur immediately before the identification column.

d. To split a nonnumeric literal between two lines, the user must ensure that the last character to be placed on the first line occurs immediately before the identification column. Otherwise, all spaces following this character on the first line will be treated as part of the literal. A hyphen is placed in the continuation column of the second line, and a quotation mark is placed at, or after, margin A, followed by the first continuing character of the literal. In cases where the last character of the literal appears immediately before the identification column on the first line and only the quotation mark that ends the literal remains to be entered, the same rule applies (a hyphen in the continuation field, a quotation mark at, or after, margin A followed by the terminating quotation mark).

Examples





e. A continuation line cannot be immediately preceded by a blank line or a comment line.

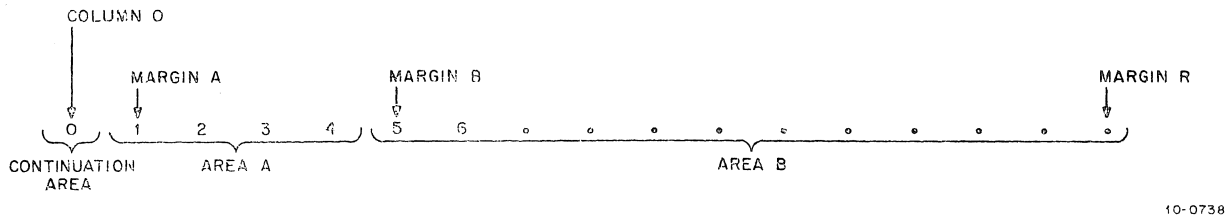
Area A occupies four character positions beginning at margin A. All division-names, section-names, and paragraph-names must begin in area A. In the DATA DIVISION, the FD and level-number entries can begin in area A, but are not required to.

Area B occupies 61 character positions beginning at margin B and ending at column 72. All remaining entries begin in area B.

The identification area occupies eight character positions beginning at the identification column and ending at margin R. This area is reserved as an identification field into which any combination of eight or fewer characters can be punched to identify the card deck. This identification is printed on a listing of the source program.

2.3.2 Non-standard Format

This format is provided for those programmers who are more familiar with the format normally used in PDP-10 operations. It differs from the standard format in that sequence numbers and identification are not used, because most PDP-10 programs do not require either. A line in non-standard format is shown below; the numbers represent character positions.



Column 0 designates a character position that is not counted by the compiler. It is only used for comment or continuation. A hyphen, asterisk, or space in column 0 is recognized by the compiler, but it is not counted by the compiler as a character position. Thus, if the user typed a space, hyphen, or asterisk in column 0 and then typed three additional spaces, he would still be in area A, not at margin B. In other words, five spaces or a horizontal tab is required to move to margin B.

Margin A designates the first character position.

Margin B designates the fifth character position relative to Margin A (not column 0). To reach margin B, the user should type horizontal tab.

Margin R designates the rightmost character position of a line.

The continuation area occupies one character position in column 0. The continuation area is used whenever it is necessary to split a word, a numeric literal, or a nonnumeric literal between the end of one line and the beginning of the next, or when a comment line is to be inserted. The following rules apply to comment or continuation lines.

a. The programmer can insert a comment line in a program by placing an asterisk (*) in the continuation area.

b. To continue a line without splitting a word or literal, the programmer must begin the first continuing word on the second line at, or after, margin A. The continuation area is left blank. As many spaces as desired can follow the last word on the first line, or the word can continue up to margin R.

c. To split a word or numeric literal from one line to the next, the programmer places a hyphen in the continuation area of the second line, and begins the first continuing character of the word or literal at or after margin A. As many spaces as desired can occur after the last character of the word or numeric literal on the first line, or the last character can occur at margin R.

d. To split a nonnumeric literal between two lines, the user must ensure that the last character to be placed on the first line occurs at margin R. Otherwise, all spaces following this character on the first line will be treated as part of the literal. A hyphen is placed in the continuation column of the second line, and a quotation mark is placed at, or after, margin A, followed by the first continuing character of the literal. In cases where the last character of the literal appears at Margin R on the first line and only the quotation mark that ends the literal remains to be entered, the same rule applies (a hyphen in the continuation field, a quotation mark at, or after, margin A followed by the terminating quotation mark).

```
                                                    ┌─Margin R
......................................................"TOTAL HOURS WORK
-"ED".....................................................
  │└─Margin A
  └──Continuation

                                                    ┌─Margin R
......................................................"TOTAL HOURS WORKED
-"".....................................................
  │└─Margin A
  └──Continuation
```

e. A continuation line cannot be immediately preceded by a blank line or a comment line.

Area A occupies four character positions beginning at margin A. All division-names, section-names, and paragraph-names must begin in area A. In the DATA DIVISION, the FD and level-number entries can begin in area A, but are not required to begin there.

Area B occupies up to 101 character positions, beginning at margin B. All remaining entries begin in area B. On an interactive terminal, the user can reach margin B by typing horizontal-tab anywhere in area A (or in column 0).

Chapter 3
The IDENTIFICATION DIVISION

The IDENTIFICATION DIVISION is required in every source program and identifies the source program and the output from compilation. In addition, the user may include other documentary information such as the name of the program's author, the name of the installation, the dates on which the program was written and compiled, any special security restrictions, and any miscellaneous remarks.

3.1 GENERAL STRUCTURE

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

3.2 TECHNICAL NOTES

a. The IDENTIFICATION DIVISION must begin with the reserved words IDENTIFICATION DIVISION followed by a period and a space.

b. The PROGRAM-ID paragraph contains the name identifying the program. The program-name may have up to six characters, and must contain only letters, digits, and the hyphen.

This paragraph must be present.

c. The remaining paragraphs are optional and, if used, may appear in any combination and in any order. A comments paragraph consists of any combination of characters from the COBOL character set organized to conform to COBOL sentence and paragraph format. All text appears as written on the output listing, except the DATE-COMPILED paragraph, which will be replaced by the current date.

Chapter 4
The ENVIRONMENT DIVISION

The ENVIRONMENT DIVISION, required in every COBOL source program, allows the programmer to describe the particular computer configurations upon which the compilation and resulting object program are to be run, and to specify the hardware features of his computer configuration.

4.1 GENERAL STRUCTURE

```
ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

[SOURCE-COMPUTER. [comment-paragraph]]

[OBJECT-COMPUTER.  PDP-10

      [MEMORY SIZE integer-1 {CHARACTERS}]
                             {WORDS     }
                             {MODULES   }

      [SEGMENT-LIMIT IS integer-2] .]

[SPECIAL-NAMES. [CONSOLE IS mnemonic-name-1]

    [CHANNEL (m) IS mnemonic-name-2

         [,CHANNEL (n) IS mnemonic-name-3]...]

    [SWITCH (m)  {IS mnemonic-name-4 [;ON STATUS IS condition-name-1]
                      [;OFF STATUS IS condition-name-2]
                 {ON STATUS IS condition-name-1
                      [;OFF STATUS IS condition-name-2]
                 {OFF STATUS IS condition-name-2
                      [;ON STATUS IS condition-name-1]
```

```
[ [SWITCH (n) .....]... ]

[CURRENCY SIGN IS literal]
[DECIMAL-POINT IS COMMA] .
```

```
INPUT-OUTPUT SECTION.

FILE-CONTROL. SELECT [OPTIONAL] file-name

    ASSIGN TO device-name-1 [,device-name-2]...

    [FOR MULTIPLE { REEL } ]
                  { UNIT }

  [ RESERVE { integer-2 } ALTERNATE [ AREA  ] ]
            { NO        }           [ AREAS ]

  [ { FILE-LIMIT IS      } [ { data-name-1 } THRU ] { data-name-2 } ]
    { FILE-LIMITS ARE    }   { literal-1   }        { literal-2   }

        [ , { data-name-3 } THRU { data-name-4 } ] ... ]
            { literal-3   }      { literal-4   }

  [ ACCESS MODE IS { SEQUENTIAL } ]
                   { RANDOM     }

    [PROCESSING MODE IS SEQUENTIAL]

    [ACTUAL KEY IS data-name-5].

    [SELECT .....]...

I-O-CONTROL. [RERUN EVERY { END OF { REEL  } } OF file-name-1]
                          {       { UNIT  } }
                          { integer-1 RECORDS }

    [SAME [RECORD] AREA FOR file-name-2, file-name-3 [, file-name-4]...]

    [MULTIPLE FILE TAPE CONTAINS file-name-5 [POSITION integer-1]
        [, file-name-6 [POSITION integer-2]] ...] .
```

4.2 CONFIGURATION SECTION

The CONFIGURATION SECTION allows the user to describe the computer configurations on which he will compile his source program and run the resulting object program. It also allows him to assign mnemonic names to certain hardware features.

```
CONFIGURATION SECTION.

[SOURCE-COMPUTER. [comment-paragraph]]

[OBJECT-COMPUTER. PDP-10
        [; MEMORY SIZE integer-1 { CHARACTERS }
                                 { WORDS      } ]
                                 { MODULES    }
        [SEGMENT-LIMIT IS integer-2].]

[SPECIAL-NAMES. [CONSOLE IS mnemonic-name-1]
    [CHANNEL (m) IS mnemonic-name-2
        [, CHANNEL (n) IS mnemonic-name-3]...]
    [SWITCH (m) { IS mnemonic-name-4 [; ON STATUS IS condition-name-1]
                        [OFF STATUS IS condition-name-2]
                { ON STATUS IS condition-name-1
                        [OFF STATUS IS condition-name-2]
                { OFF STATUS IS condition-name-2
                        [ON STATUS IS condition-name-1] }
        [SWITCH (n) ......]...]
    [CURRENCY SIGN IS literal]
    [DECIMAL-POINT IS COMMA] .]
```

Technical Notes

a. This section must appear in every source program.

b. All commas and semicolons are optional. A period must terminate the entire entry in each of the three paragraphs.

SOURCE-COMPUTER

Function

The SOURCE-COMPUTER paragraph describes the computer on which the program is to be compiled.

General Format

```
[SOURCE-COMPUTER.  [comment-paragraph.]]
```

Technical Notes

a. This paragraph is optional.

b. This paragraph is for documentation only. The comment paragraph is replaced in the listing by the word PDP-10.

OBJECT-COMPUTER

Function

The OBJECT-COMPUTER paragraph describes the computer on which the program is to be executed.

General Format

```
OBJECT-COMPUTER. PDP-10

[ MEMORY SIZE integer { CHARACTERS } ]
                      { WORDS      }
                      { MODULES    }
[SEGMENT-LIMIT IS integer-2] .
```

Technical Notes

a. This paragraph is optional.

b. PDP-10 must appear as the first entry following the paragraph-name OBJECT-COMPUTER.

c. The MEMORY SIZE clause is optional. If it is omitted, 262,144 WORDS are assumed. If it appears, the following ranges are applicable.

CHARACTERS	Up to 1,572,864 (262,144 words x 6 character/word)
WORDS	Up to 262,144
MODULES	Up to 256 (1 module equals 1024 words)

The MEMORY SIZE clause indicates the amount of core for the object code only, and does not include the core required by the COBOL operating system.

d. If the SEGMENT-LIMIT clause is given, only those segments having priority numbers from 0 up to, but not including, the value of integer-2 are considered as resident segments of the program. Integer-2 must be a positive integer in the range 1 to 49.

If the SEGMENT-LIMIT clause is omitted, segments having priority numbers from 0 through 49 are considered as resident segments of the program (that is, SEGMENT-LIMIT IS 50 is assumed).

More on segmentation can be found in Chapter 6.

SPECIAL-NAMES

Function

The SPECIAL-NAMES paragraph provides a means of associating hardware devices with user-specified mnemonic names.

General Format

```
SPECIAL-NAMES. [CONSOLE IS mnemonic-name-1]
    [CHANNEL (m) IS mnemonic-name-2
        [CHANNEL (n) IS mnemonic-name-3]...]

    [SWITCH (m) { IS mnemonic-name-4 [; ON STATUS IS
                  ON STATUS IS condition-name-1 [; OFF
                  OFF STATUS IS condition-name-2 [; ON

                      condition-name-1] [; OFF STATUS IS condition-name-2] }
                  STATUS IS condition-name-2]
                  STATUS IS condition-name-1]
        [SWITCH (n).....]..]

    [CURRENCY SIGN IS literal]

    [DECIMAL-POINT IS COMMA] .
```

Technical Notes

a. This paragraph is optional.

b. The name CONSOLE refers to the user's Teletype console. The assigned mnemonic-name may be used with the ACCEPT and DISPLAY verbs in the PROCEDURE DIVISION to input data from and output data to the console.

c. The name CHANNEL refers to a channel on the line-printer control tape. m and n represent any integer from 1 to 8 and refer to any one of the eight channels on the tape. Control tape channels can be referred to in the ADVANCING clause of the WRITE verb in the PROCEDURE DIVISION to advance the paper form to the desired channel position. For example, if the entry

CHANNEL (1) IS TOP-OF-PAGE

is included in this paragraph, the following procedure statement will print the line and then skip to the top of the next page.

IF LINE-COUNTER IS GREATER THAN 50 WRITE PRINT-RECORD BEFORE ADVANCING TOP-OF-PAGE.

d. The name SWITCH refers to the hardware switches on the PDP-10 console. m and n represent any integer from 0 to 35 and refer to the corresponding console switches.

The mnemonic-name can be used in conditional expressions in the PROCEDURE DIVISION. For example, if the entry

SWITCH (4) IS INPUT-1

is included in this paragraph, the following condition is considered to be true if switch (4) is on.

IF INPUT-1 IS ON....

If a condition-name is specified for the ON or OFF STATUS of a switch, that condition-name can be used in a conditional expression. For example, if the entry

SWITCH (4) IS INPUT-1; OFF STATUS IS NO-INPUT

is included in this paragraph, the following procedure statements are functionally equivalent.

IF INPUT-1 IS OFF....

IF NO-INPUT....

e. The literal which appears in the CURRENCY SIGN clause must be used in PICTURE clauses (DATA DIVISION) to represent the currency symbol. If this clause is not present, only the standard $ may be used as a currency symbol in a PICTURE clause.

This literal is limited to a single printable character and must not be one of the following characters:

digits 0 through 9

alphabetic characters A, B, C, D, P, R, S,V, X, Z

special characters * + - , . ; ( ) "

f. The clause DECIMAL-POINT IS COMMA, if present, causes the functions of the comma and the period to be interchanged in any PICTURE clause character-string and in any numeric literal.

4.3 INPUT-OUTPUT SECTION

The INPUT-OUTPUT SECTION names the files and external media required by the object program and provides information required for transmission and handling of data during execution of the object program.

```
INPUT-OUTPUT SECTION.
FILE-CONTROL. SELECT [OPTIONAL] file-name
    ASSIGN TO [integer-1] device-name-1 [, device-name-2]
    [ FOR MULTIPLE { REEL } ]
                   { UNIT }
    [ RESERVE { integer-2 } ALTERNATE [ AREA ] ]
              { NO        }           [ AREAS ]
    [ { FILE-LIMIT IS  } [ { data-name-1 } THRU ] { data-name-2 } ]
      { FILE-LIMITS ARE }  { literal-1   }       { literal-2   }
        [ , { data-name-3 } THRU { data-name-4 } ]...
            { literal-3   }      { literal-4   }
    [ ACCESS MODE IS { SEQUENTIAL } ]
                     { RANDOM     }
    [ PROCESSING MODE IS SEQUENTIAL ]
    [ ACTUAL KEY IS data-name-5 ] .
    [ SELECT....] ...

I-O-CONTROL.
    [ RERUN EVERY { END OF { REEL } } OF file-name-1 ]
                  {        { UNIT } }
                  { integer-1 RECORDS }
```

```
[ [ [SAME [RECORD] AREA FOR file-name-2, file-name-3 [ , file-name-4] ...]
  [ [MULTIPLE FILE TAPE CONTAINS file-name-5 [POSITION integer-2]
    [, file-name-6 [POSITION integer-3]]...] ] ]
```

Technical Notes

a. This section is optional.

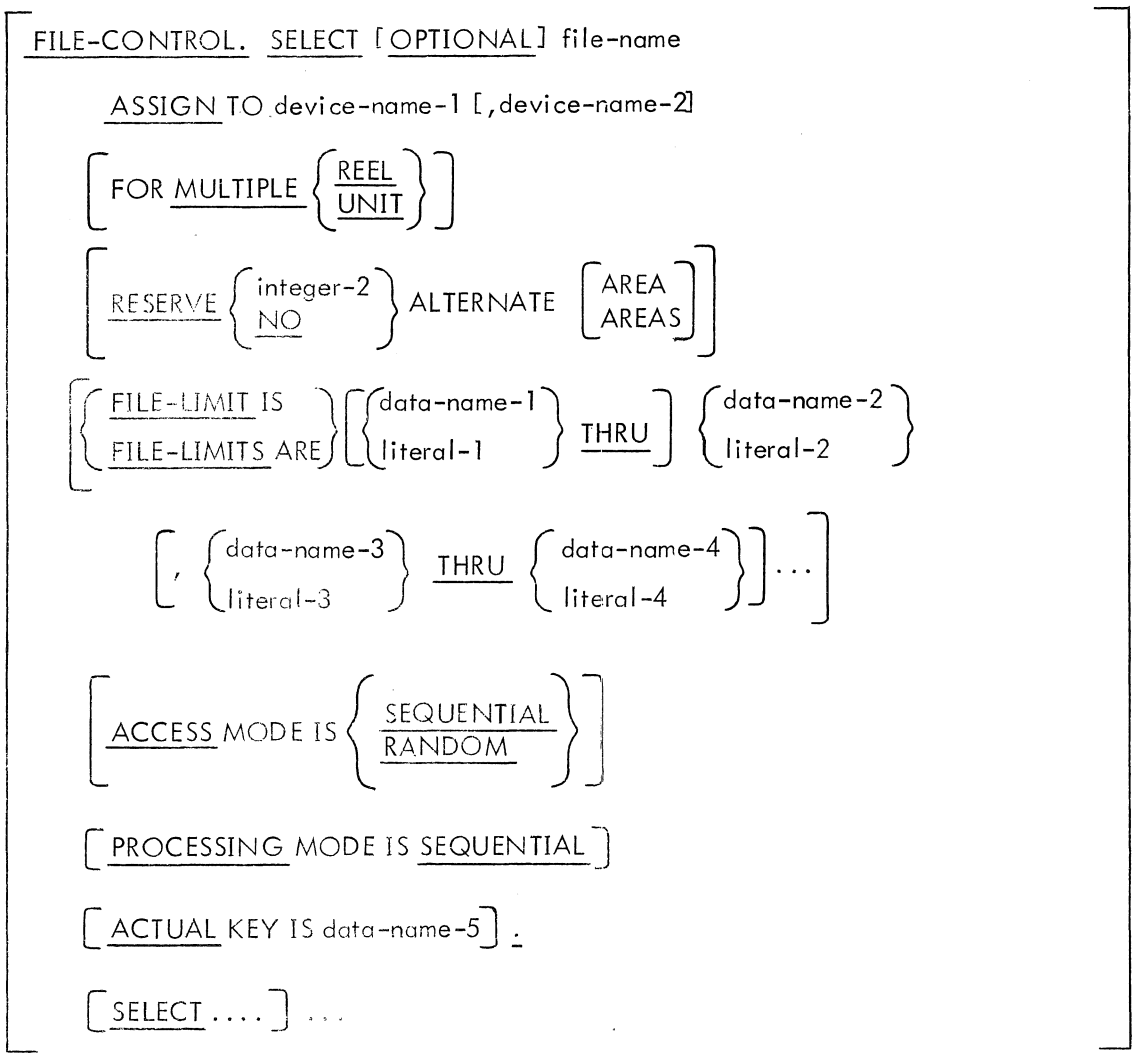
b. All semicolons and commas are optional. Each SELECT statement in the FILE-CONTROL paragraph must end with a period. The entire entry in the I-O-CONTROL paragraph must end with a period.

FILE-CONTROL

Function

The FILE-CONTROL paragraph names each file, identifies the file medium, and allows hardware assignments.

General Format



Technical Notes

a. Each file described in the DATA DIVISION must be named once and only once as a file-name in a SELECT statement. Conversely, each file named in a SELECT statement must have a File Description entry in the DATA DIVISION. Each file-name must be unique within a program.

b. The key word OPTIONAL is required for input files that are not necessarily present each time the object program is run. When an OPEN statement is executed for a file that has been declared OPTIONAL, the question "IS file-name PRESENT?" is typed and the operator responds with "YES" or "NO". If the response is "YES", the file is processed normally; if the response is "NO", the first READ statement executed for the file will immediately take the AT END or INVALID KEY path.

c. The ASSIGN clause specifies the file medium. Device-names may be either physical device-names or logical device-names.

Physical device-names are fixed mnemonic-names that are associated with specific peripheral devices. When specified in an ASSIGN clause, a physical device-name assigns the associated file to that device. Physical device names presently employed are:

DSK	for the disk
TTY	for the Teletype

Since Batch uses a phenomenen known as the pseudo-Teletype for input and output, users must assign card files to TTY, and printer files to TTY, ensuring that these files are not open at the same time in the program.

More than one device may be assigned to a file to avoid delay when switching from one reel or unit to the next. When more than one device is specified, the object program automatically uses the next device, in a cyclical manner, when an end-of-reel condition is detected.

Whether or not multiple devices are assigned, the FOR MULTIPLE $\left\{\begin{matrix}\text{REEL}\\\text{UNIT}\end{matrix}\right\}$ clause must be included for any file that occupies (or might occupy) more reels than the number of devices assigned.

d. The RESERVE clause allows the user to specify the number of input-output buffer areas to be allocated by the compiler to this file.

If the access mode is RANDOM, this clause is ignored, and only one buffer area is assigned.

If the NO option is used, only two such areas will be allocated.

If the integer-2 option is used, the integer specifies the number of areas to be assigned in addition to the two areas always assigned by the compiler. Integer-2 must be positive.

e. The FILE-LIMIT clause is required only for files whose access mode is RANDOM; it is optional for files with SEQUENTIAL access mode residing on mass-storage devices; it is ignored in all other cases.

Each pair of operands within this clause represents a logical portion of the file. If the first of a pair of operands is not specified, it is assumed to be 1.

The operands represent logical record numbers relative to the beginning of the file.

The logical beginning of a random-access file is considered to be that record represented by the first operand of the FILE-LIMIT clause. The logical end of a random-access file is considered to be that record represented by the last operand.

The value of data items specified in this clause is utilized by the object operating system only when the file is opened by an OPEN procedure statement.

f. The ACCESS MODE clause is required only for random-access files; it is ignored in all other cases.

If ACCESS MODE IS SEQUENTIAL, the random-access records are obtained or placed sequentially. That is, the next logical record is made available from the file on a READ statement execution, and an output record is placed into the next available area on a WRITE statement execution. Thus, sequential access processing on a random-access device is functionally similar to the processing of a magnetic tape file.

If ACCESS MODE IS RANDOM, the contents of the data item associated with the ACTUAL KEY specifies which record, relative to the beginning of the file, is made available by a READ statement, or where the record is to be placed by a WRITE statement.

g. PROCESSING MODE IS SEQUENTIAL is for documentation only; records are always processed in the order in which they are accessed.

h. If a sort-name is selected, at least three devices must be assigned. The devices must be retrievable, i.e., disks, DECtapes, or magnetic tapes.

i. The ACTUAL KEY data item must be defined as a level-77 COMPUTATIONAL item, the PICTURE of which contains only the characters 5 and 9.

I-O-CONTROL

Function

The I-O-CONTROL paragraph specifies the points at which a rerun dump is to be performed, the memory area which is to be shared by different files, and the location of files on a multiple-file reel.

General Format

```
I-O-CONTROL.
    [ RERUN EVERY { END OF { REEL } } OF file-name-1 ]
                  {       { UNIT } }
                  { integer-1 RECORDS }
    [ SAME [ RECORD ] AREA FOR file-name-2, file-name-3 [, file-name-4] ... ]
    [ MULTIPLE FILE TAPE CONTAINS file-name-5 [ POSITION integer-2 ]
        [, file-name-6 [ POSITION integer-3 ] ... ] ... . ]
    [ SEGMENT-LIMIT IS integer-4 ] .
```

Technical Notes

a. This paragraph is optional.

b. The RERUN clause specifies when a rerun dump is to be performed.

The dump is always written onto a disk file, using the program-name (from the PROGRAM-ID paragraph in the IDENTIFICATION DIVISION) as the filename, and an extention of RER.

If the END OF $\left\{\begin{matrix}\text{UNIT}\\\text{REEL}\end{matrix}\right\}$ option is used, a rerun dump is taken at the end of each input or output reel of the specified file.

If the integer-1 RECORDS option is used, a rerun dump is taken whenever a number of logical records equal to a multiple of integer-1 is either read or written for the file.

c. The SAME AREA clause specifies that two or more files are to use the same area during processing; this includes all buffer areas and the record area.

If the RECORD option is specified, the files share only the record area (i.e., the area in which the current logical record is processed). If the RECORD option is not used, only one of the named files can be open at one time.

d. The MULTIPLE FILE clause is required when more than one file shares the same physical reel of tape. This clause is invalid for media other than magnetic tape.

Regardless of the number of files on a single reel, only those files defined in the program may be listed. If all files residing on the tape are listed in consecutive order, the POSITION option need not be given. If any file on the tape is not listed, the POSITION option must be included; integer-2, integer-3, etc., specify the position of the file relative to the beginning of the tape.

All files on the same reel of tape must be ASSIGNED to the same device in the FILE-CONTROL paragraph.

Not more than one file on the same reel of tape can be open at one time.

Chapter 5
The DATA DIVISION

The DATA DIVISION, required in every COBOL program, describes the characteristics of the data to be processed by the object program.

This data can be divided into two major types:

a. Data contained in files, both input and output.

b. Data initially stored as part of the program (e.g., constant data such as messages, tables of fixed values, etc.), or data developed during processing (e.g., intermediate information such as partial arithmetic results).

To handle these two types of data, the DATA DIVISION consists of two sections:

a. The FILE SECTION, which describes the characteristics and the data formats for each file processed by the object program.

b. The WORKING-STORAGE SECTION, which contains any fixed values and the working areas into which intermediate data can be stored.

5.1 FILE SECTION

In the FILE SECTION, the characteristics of each file to be processed are described by two types of entries.

The first type of entry, the file description (FD), describes the physical aspects of the file. These aspects include

a. How the logical data records of the file are physically grouped into blocks on the file medium

b. The maximum length of a logical record

c. Whether or not the file contains header and trailer labels and, if so, whether the format of these labels is standard or nonstandard

d. The names of the records contained in the file.

The second type of entry, the data description, describes the data formats of the logical records in the files.

The FILE SECTION begins with the section-header FILE SECTION.

5.2 WORKING-STORAGE SECTION

The WORKING-STORAGE SECTION is used to define (1) data (such as constant values and messages) that is to be initially stored when the object program is loaded and (2) areas that are to be used for storing intermediate results. The WORKING-STORAGE SECTION is similar to the FILE SECTION, except that (1) it cannot contain FD and SD entries and (2) it may contain level-77 entries.

The WORKING-STORAGE SECTION begins with the section-header WORKING-STORAGE SECTION.

5.3 DATA DESCRIPTIONS

5.3.1 Elementary Items and Group Items

The basic user-defined datum in a COBOL program is called an elementary item; it may be referenced directly only as a unit. An elementary item may be associated with contiguous elementary items to form blocks of data; these blocks are called group items. Group items may be associated with other group items and/or elementary items to form more inclusive group items. Thus an elementary item may be contained within more than one group item, and a group item may contain more than one elementary item.

5.3.2 Independent Items and Nonindependent Items

Items fall into one of two classes: independent items and nonindependent items. Independent items are those that are not contained within any other data item; independent items may be either group items or elementary items. Nonindependent items are those contained within some other item; nonindependent items may be either group or elementary items. Every nonindependent item must belong to some independent item.

5.3.2.1 Level Numbers - The level number indicates the class to which a data item belongs. Independent items must be assigned a level number of 2 through 49, or the special level number 66. Independent items with a level number of 1 are called records; they may contain nonindependent items.

Independent items with a level number of 77 are called noncontiguous elementary items; they may not contain nonindependent items (hence, level 77 items must be elementary items).

The sequence of level numbers in definitional entries is the means of determining the association of items into blocks. Each independent item is a block; the independent item ends where another independent item begins, signified by the occurrence of a level number of 1 or 77, or when the special level number 66 or a section or division header, or one of the special level indicators FD or SD is encountered. A nonindependent item is a block that ends when another item of the same or a numerically lower level number is encountered, or when the independent item containing it is terminated (as determined by the criteria stated above).

Level-66 data items are special blocks that contain an explicitly specified portion of a record (or possibly the whole record). These items are explained in detail under the RENAMES clause.

5.3.3 Records and Files

Records may be divided into two categories: those associated with a file and those not associated with a file. A file is the highest level of data organization in COBOL; and in PDP-10 COBOL, a file represents a collection† of data held on some external medium, i.e., not wholly in real or virtual core storage.

5.4 QUALIFICATION

Any data item which is to be referenced must be uniquely identified. This can be achieved by assigning a unique name to each item; however, in many applications this is tedious and inconvenient (1) because of the large number of names required and (2) because items containing the same type of information on different blocks would have different names. Therefore, qualification is introduced to allow similar nonindependent items and certain records to have identical names.

Qualification is simple; it means giving enough information about the item to specify it uniquely. In COBOL this information is the names of the blocks containing it, in order of increasing inclusiveness. It is not necessary to name each block containing it but only enough blocks so that no other item with the same name as the original item could be identically qualified. It is also unnecessary to name each successively higher block containing the item until a unique qualification is made; any set of block names that uniquely describe the item may be used.

---

†A collection may be thought of as a block.

Example:

```
01        RECORD-1.                    01        RECORD-2.
  02        ITEM-1.                      02        ITEM-2.
    03        SUB-ITEM.                    03        SUB-ITEM.
      04        FIELD PIC X.                 04        FIELD PIC X.
```

FIELD in the left-hand example can be referenced uniquely in any of the following ways:

```
FIELD OF SUB-ITEM OF ITEM-1 OF RECORD-1.
FIELD OF SUB-ITEM OF ITEM-1.
FIELD OF SUB-ITEM IN RECORD-1.
FIELD IN ITEM-1 OF RECORD-1.
FIELD IN RECORD-1.
FIELD IN ITEM-1.
```

Since the connectives OF and IN are equivalent, they may be used interchangeably.

The only data items which need have unique names are level-77 items and records not associated with files, since they are not contained in any higher level data structure. Records associated with files may be qualified by the file name, as may any item contained within the record. File names must be unique.

Level-66 items may be qualified only (1) by the name of the record with which they are associated and (2) by the name of any file with which that record is associated.

5.5 SUBSCRIPTING AND INDEXING

In some applications, it is convenient to specify a block of data as consisting of repetitions of some block of data. The COBOL facility for this function is the OCCURS clause (described below), which specifies that the block of data in which the clause appears actually consists of some user-specified number of repetitions of the same block. This does not mean that the block is a group item containing the repetitions, but rather that the block itself, regardless of whether it is a group or elementary item, is repeated and that any block containing this block actually contains all of the identical blocks.

Since each of these repeated blocks has the same name as the originally described block and is at exactly the same place in the same hierarchy, qualification is not sufficient to specify which of these blocks one wishes to reference; subscripting (or indexing, which is identical to subscripting in PDP-10 COBOL) must be used to distinguish among the repetitions.

Subscripting is merely a formal method of specifying which repetition of the block is desired and must be used whenever reference is made to any block described with an OCCURS clause or to any item

contained within such a block, since each of the latter also is part of each of the repetitions. Thus, if an OCCURS clause appears in an entry contained in a block with an OCCURS clause, it is necessary to specify (1) in which of the repetitions of the more inclusive block the desired item can be found, and then (2) which of the repetitions of the less inclusive block within the particular repetition of the more inclusive block, is desired. The format for subscripting is described in the OCCURS clause.

FILE DESCRIPTION (FD)

Function

The File Description (FD) furnishes information concerning the physical structure, identification, and record names pertaining to a given file.

General Format

```
FD  file-name
    [ BLOCK CONTAINS [integer-1 TO] integer-2 { RECORDS / CHARACTERS } ]
    [ RECORD CONTAINS [integer-3 TO] integer-4 CHARACTERS ]
    [ LABEL { RECORD IS / RECORDS ARE } { STANDARD / OMITTED / record-name-1 [, record-name-2] ... } ]
    [ VALUE OF { IDENTIFICATION / DATE-WRITTEN } IS { data-name-1 / literal-1 }
      [ , { DATE-WRITTEN / IDENTIFICATION } IS { data-name-2 / literal-2 } ] ]
      ...
    [ DATA { RECORD IS / RECORDS ARE } record-name-3 [ , record-name-4 ] ... ] .
```

Technical Notes

a. An FD entry must be present for each file-name selected in the FILE-CONTROL paragraph of the ENVIRONMENT DIVISION.

b. All semicolons and commas are optional. The entire FD entry must terminate with a period.

c. The clauses may appear in any order within the File Description entry.

d. Each of the above clauses appears in alphabetical order on the following pages.

BLOCK CONTAINS

Function

The BLOCK CONTAINS clause specifies the size of a physical block.

General Format

```
[ BLOCK CONTAINS [integer-1 TO] integer-2 { RECORD(S) / CHARACTERS } ]
```

Technical Notes

a. If this clause is not present, the records will be read or written without regard to any physical boundaries imposed by the device.

b. If the CHARACTERS option is used, the physical block size is specified in terms of the number of character positions required to contain the record. If the recording mode is ASCII (that is, all records for the file are described, explicitly or implicitly, as USAGE DISPLAY-7), it is assumed that the size is specified in terms of DISPLAY-7 characters. If the recording mode is SIXBIT (that is, the records for the file are all described as other than USAGE DISPLAY-7), it is assumed that the size is specified in terms of DISPLAY-6 characters. See "USAGE" for a discussion of the USAGE of group items, including records.

c. Integer-1 and integer-2 must be positive integers. If only integer-2 is specified, it represents the exact size of the physical block. If both integer-1 and integer-2 are given, they represent the minimum and maximum sizes of the physical block.

DATA RECORD

Function

The DATA RECORD clause cross references the record descriptions with their associated file.

General Format

```
DATA { RECORD IS  }  record-name-3  [ , record-name-4 ]  ...
     { RECORDS ARE }
```

Technical Notes

a. This clause is optional. If this clause is not included, all records not associated with a LABEL RECORDS clause will be assumed to be data records.

b. Both record-name-3 and record-name-4 must be the names given in 01-level data entries subordinate to this FD. The presence of more than one such record-name indicates that the file contains more than one type of data record. These records may have different descriptions. The order in which they are listed is not significant.

c. All records within a file share the same area.

FD file-name

Function

The FD file-name clause identifies the file to which this file description entry and the subsequent record descriptions relate.

General Format

FD file-name

Technical Notes

a. This entry must begin each file description.

b. The file-name must appear in a SELECT statement in the FILE-CONTROL paragraph of the ENVIRONMENT DIVISION.

LABEL RECORD

Function

The LABEL RECORD clause specifies whether or not labels are present on the file and, if so, identifies the format of the labels.

General Format

```
[ LABEL { RECORD IS  }  { OMITTED                                  }  ]
        { RECORDS ARE }  { STANDARD                                }
                         { record-name-1[, record-name-2] . . .    }
```

Technical Notes

a. If the clause is omitted, LABEL RECORDS ARE STANDARD is assumed.

b. The OMITTED option is used when the file has no header or trailer labels.

c. The STANDARD option is used when the file has header and trailer labels that conform to the PDP-10 standard format (see Chapter 8). LABEL RECORDS ARE STANDARD must be specified for files on disk

d. The record-name option is used when the file labels do not conform to the PDP-10 standard format. The record-names must appear as the name of a record description (level-01) subordinate to this FD; the record-names must not appear in a DATA RECORDS clause.

RECORD CONTAINS

Function

The RECORD CONTAINS clause specifies the size of the data records in this file.

General Format

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

Technical Notes

a. Because the size of each data record is completely defined by its record description entry, this clause is for documentation purposes only and is never required. However, if it is used, the following rules must be observed.

b. Integer-1 and integer-2 must be positive integers. Integer-2 may not be less than the size of the largest record.

c. The data record size is specified in terms of the number of character positions required to contain the record.

SD file-name

Function

The SD file-name clause identifies the sort file to which this file description entry and the subsequent record descriptions relate.

General Format

SD file-name

Technical Notes

a. This entry must begin each sort file description.

b. The file-name must appear in a SELECT statement in the FILE-CONTROL paragraph of the ENVIRONMENT DIVISION.

c. The DATA RECORD and RECORD CONTAINS clauses are the only descriptive clauses allowed.

VALUE OF IDENTIFICATION/DATE-WRITTEN

Function

The VALUE OF clause provides specific data for an item within the label records associated with a file.

General Format

```
[ VALUE OF { IDENTIFICATION  } IS { data-name-1 }
           { DATE-WRITTEN    }     { literal-1   }

    [, { DATE-WRITTEN  } IS { data-name-2 } ]
       { IDENTIFICATION }   { literal-2   }
]
```

Technical Notes

a. The VALUE OF IDENTIFICATION clause is required only if label records are standard; it is ignored in all other cases. The VALUE OF DATE-WRITTEN is always optional.

b. Only one value may be specified for IDENTIFICATION, and only one value for DATE-WRITTEN, for each file.

c. IDENTIFICATION represents the file-name and file-name extension of a file with standard labels. If a data-name is specified, it must be associated with a data item nine characters in length. If a literal is specified, it must be a nonnumeric literal nine characters in length. The first six characters are taken as the file-name, and last three characters are taken as the extension. The programmer must provide spaces as required to conform to this convention.

Examples:

(1) VALUE OF IDENTIFICATION IS "COST ΔΔTST"

(2) VALUE OF IDENTIFICATION IS FILE-1-NAME

```
        .
        .
(WORKING-STORAGE SECTION.)
        .
        .
77 FILE-1-NAME PICTURE IS X(9).
```

d. DATE-WRITTEN represents the date that a file (with STANDARD labels) was written. If a data-name is specified, it must be associated with a data item six characters in length. If a

literal is specified, it must be a nonnumeric or numeric literal six characters in length. The first two characters are taken as year, the next two as month, and the last two as day. The DATE-WRITTEN clause is ignored when the file is OPENed for output; instead, the current date is used.

Examples:

(1) VALUE OF IDENTIFICATION IS "RANDOMXYZ", DATE-WRITTEN IS 690612

(2) VALUE OF IDENTIFICATION IS "DATA$\Delta\Delta\Delta\Delta\Delta$", DATE-WRITTEN IS FILE-1-DATE

```
.
.
(WORKING-STORAGE SECTION.)
 77 FILE-1-DATE PICTURE IS 9(6).
```

e. For input files, this information is checked against the file when the file is OPENed. For output files, the VALUE OF IDENTIFICATION is written when the file is OPENed. See Chapter 8, "Standard Label Procedures".

f. All data items must be either level-77 or level-01 items in working-storage, with usage either DISPLAY-6 or DISPLAY-7.

RECORD DESCRIPTIONS

Following the FD for a file, a record description is given for each different record format in the file. A record description begins with a level-01 entry:

```
01  data-name
```

where the data-name is one of those listed in the DATA RECORDS clause of the FD.

A complete record description may be as simple as

```
01  data-name  PICTURE picture-string.
```

or it may be more complex, where the 01-level is followed by a long series of data description entries of varying hierarchies that describe various portions and subportions of the record.

Record Concepts

A record description consists of a set of data description entries which describe a particular logical record. Each data description entry consists of a level-number followed by a data-name (or FILLER) which is followed, as required, by a series of descriptive clauses.

The general format of a data description entry follows.

DATA DESCRIPTION ENTRY

Function

A data description entry describes a particular item of data.

General Format

```
level-number { data-name-1 }
             { FILLER      }

[ REDEFINES data-name-2 ]

[ { PICTURE } IS  picture-string ]
[ { PIC     }                    ]

[                                                  ]
[ USAGE IS { COMPUTATIONAL   }                     ]
[          { COMP            }                     ]
[          { COMPUTATIONAL-1 }                     ]
[          { COMP-1          }                     ]
[          { DISPLAY         }                     ]
[          { DISPLAY-6       }                     ]
[          { DISPLAY-7       }                     ]
[          { INDEX           }                     ]
[                                                  ]

[ { SYNCHRONIZED } { LEFT  } ]
[ { SYNC         } { RIGHT } ]

[ { JUSTIFIED } { RIGHT } ]
[ { JUST      } { LEFT  } ]

[ BLANK WHEN ZERO ]

[ VALUE IS literal-1 ]

[ OCCURS [ integer-1 TO ] integer-2 TIMES [ INDEXED BY index-name-1 [ , index-name-2 ]... ]
                                          [                                               ]
        [ DEPENDING ON data-name-1 ]      [                                               ]
```

RENAMES ENTRY

```
66 data-name-1 RENAMES data-name-2 [THRU data-name-3] .
```

CONDITION-NAME ENTRY

```
88 condition-name {VALUE IS  } literal-1 [THRU literal-2]
                  {VALUES ARE}

   [, literal-3 [THRU literal-4]] ... .
```

Technical Notes

a. Each data description entry must be terminated by a period. All semicolons and commas are optional.

b. The clauses may appear in any order, with one exception: the REDEFINES clause, when used, must immediately follow the data-name.

c. The VALUE clause must not appear in a data description entry which also contains an OCCURS clause, or in an entry which is subordinate to an entry containing an OCCURS clause. The latter part of this rule does not apply to condition-name (level-88) entries.

d. The PICTURE clause must be specified for every elementary item, except a USAGE INDEX or COMP-1 item.

e. The clauses SYNCHRONIZED, PICTURE, JUSTIFIED, and BLANK WHEN ZERO can be specified only at the elementary level.

f. The clauses shown in the General Format appear in alphabetical order on the following pages.

BLANK WHEN ZERO

Function

The BLANK WHEN ZERO clause causes the blanking of an item when its value is zero.

General Format

[BLANK WHEN ZERO]

Technical Notes

a. When the BLANK WHEN ZERO option is used and the item is zero, the item is set to blanks.

b. BLANK WHEN ZERO can be specified only at the elementary level and only for numeric or numeric-edited items whose usage is DISPLAY-6 or DISPLAY-7.

c. More comprehensive editing features are available in the PICTURE clause. If a PICTURE clause appears in the same data description entry and contains the zero suppression symbol * (zero suppress and replace with *), the field is replaced with * (see PICTURE).

condition-name (level-88)

Function

To assign a name to a value or range of values of the associated data item.

General Format

```
88 condition-name {VALUE IS  } literal-1 [THRU literal-2]
                  {VALUES ARE}

   [, literal-3 [THRU literal-4]] ... .
```

Technical Notes

a. Each condition-name requires a separate level-88 entry. This entry contains the name assigned to the condition, and the value or values associated with that condition. Condition-name entries must immediately follow the data description entry with which the condition-name is to be associated.

b. A condition-name entry can be associated with any elementary or group item except

(1) another condition-name entry, or

(2) a level-66 item.

c. Some examples of possible level-88 entries are given below.

(1)
```
05 B-FIELD PICTURE IS 99.
   88 B1 VALUE IS 3.
   88 B2 VALUES ARE 50 THRU 69.
   88 B3 VALUES ARE 20, 25, 28, 31 THRU 37.
   88 B4 VALUES ARE 70 THRU 75, 80 THRU 85, 90 THRU 95.
```

(2)
```
02 C-FIELD PICTURE IS XXX.
   88 C-YES VALUE IS "YES".
   88 C-NO VALUE IS "NOΔ".
```

d. The data item with which the condition-name is associated is called a conditional variable. A conditional variable may be used to qualify any of its condition-names. If references to a conditional variable require indexing, subscripting, or qualification, then references to its associated condition-names also require the same combination of indexing, subscripting, or qualification.

e. A condition-name is used in conditional expressions as an abbreviation for the related condition. Thus, if the above DATA DIVISION entries (note c) are used, the statements in each pair below are functionally equivalent.

	Relational Expression	Condition-Name
(1)	IF B-FIELD IS EQUAL TO 3	IF B1 ....
(2)	IF B-FIELD IS GREATER THAN 49 AND LESS THAN 70	IF B2 ....
(3)	IF B-FIELD IS EQUAL TO 20 OR EQUAL TO 25 OR EQUAL TO 28 OR GREATER THAN 30 AND LESS THAN 38	IF B3 ....
(4)	IF B-FIELD IS GREATER THAN 69 AND LESS THAN 76 OR GREATER THAN 79 AND LESS THAN 86 OR GREATER THAN 89 AND LESS THAN 96	IF B4 ....
(5)	IF C-FIELD IS EQUAL TO "YES"	IF C-YES ....

f. Literal-1 must always be less than literal-2, and literal-3 less than literal-4. The values given must always be within the range allowed by the format given for the conditional variable. For example, any condition-name values given for a conditional variable with a PICTURE of PP999 must be in the range of .00000 to .00999. (See Note j under PICTURE in this chapter for the meaning of P in a picture-string.)

data-name/FILLER

Function

A data-name specifies the name of the data being described. The word FILLER specifies an unreferenced portion of the logical record.

General Format

level-number { data-name / FILLER }

Technical Notes

a. A data-name or the word FILLER must immediately follow the level-number in each data description entry.

b. A data-name must be composed of a combination of the characters A through Z, 0 through 9, and the hyphen. It must contain at least 1 alphabetic character and must not exceed 30 characters in length. It must not duplicate a COBOL reserved word.

c. The key word FILLER is used to name an unreferenced elementary item in a record (that is, an item to which the programmer has no reason for assigning a unique name). A FILLER item cannot, under any circumstances, be referenced directly in a PROCEDURE DIVISION statement. However, it may be indirectly referenced by referring to a group-level item of which the FILLER item is a part.

JUSTIFIED

Function

The JUSTIFIED clause specifies nonstandard positioning of data within a receiving data item.

General Format

```
[ {JUSTIFIED} {RIGHT} ]
  {JUST     } {LEFT  }
```

Technical Notes

a. The JUSTIFIED clause cannot be specified at a group level, or for numeric or numeric edited items. If neither RIGHT nor LEFT is specified, RIGHT is assumed.

b. The standard rules for positioning data within an elementary data item are as follows:

(1) Receiving data item described as numeric or numeric-edited (see definitions in Notes f and i under PICTURE in this chapter).

The data is aligned by decimal point and is moved to the receiving character positions with zero fill or truncation on either end as required.

If an assumed decimal point is not explicitly specified, the data item is treated as if it had an assumed decimal point immediately following its rightmost character, and the sending data is aligned according to this decimal point.

(2) Receiving data item described as alphanumeric (other than numeric edited) or alphabetic (see definitions in Notes e and g under PICTURE in this chapter).

The data is moved to the receiving character positions and aligned at the leftmost character position with space fill or truncation at the right end as required.

c. When a receiving item is described as JUSTIFIED LEFT, positioning occurs as in b (2) above.

d. When a receiving data item is described with the JUSTIFIED RIGHT clause and is larger than the sending data item, the data is aligned at the rightmost character position in the receiving item with space fill at the left end.

When a receiving data item is described with the JUSTIFIED RIGHT clause and is smaller than the sending data item, the data is aligned at the rightmost character position in the receiving item with truncation at the left end.

Examples are given below.

```
03  ITEM-A  PICTURE IS
    X(8) VALUE IS "ABCDEFGH".
03  ITEM-B  PICTURE IS
    X(4) VALUE IS "WXYZ".
03  ITEM-C  PICTURE IS X(6).
03  ITEM-D  PICTURE IS X(6)
    JUSTIFIED RIGHT.
```

	Contents of Receiving Field
MOVE ITEM-A TO ITEM-C.	A B C D E F
MOVE ITEM-A TO ITEM-D.	C D E F G H
MOVE ITEM-B TO ITEM-C.	W X Y Z Δ Δ
MOVE ITEM-B TO ITEM-D.	Δ Δ W X Y Z

level-number

Function

The level-number shows the hierarchy of data within a logical record. In addition, special level-numbers are used for condition-names (level-88), noncontiguous WORKING-STORAGE items (level-77), and the RENAMES clause (level-66).

General Format

level-number { data-name / FILLER }

Technical Notes

a. A level-number is required as the first element in each data description entry.

b. Level-numbers may be placed anywhere on the source line, at or after margin A.

c. Level-number 88 is described under "condition-name (level-88)", and level-number 66 is described under "RENAMES (level-66)", both in this section.

d. A further description of level-numbers and data hierarchy can be found in the introduction to this chapter.

OCCURS

Function

The OCCURS clause eliminates the need for separate entries for repeated data and supplies information required for the application of subscripts and indexes.

General Format

```
[ OCCURS [integer-1 TO] integer-2 TIMES [INDEXED BY index-name-1 [, index-name-2]...] ]
                [DEPENDING ON data-name-1] ]
```

Technical Notes

a. This clause cannot be specified in a data description entry that has a 66 or 88 level-number, or in one that contains a VALUE clause.

b. The OCCURS clause is used in defining tables or other homogeneous sets of repeated data. Whenever this clause is used, the associated data-name and any subordinate data-names must always be subscripted or indexed when used in a PROCEDURE DIVISION statement.

c. All clauses given in a data description entry that includes an OCCURS clause apply to each repetition of the item.

d. The integers must be positive. If integer-1 is specified, it must have a value less than integer-2. No value of a subscript may exceed integer-2; in addition, if data-name-1 is specified, no subscript may exceed the value of data-name-1 at the time of subscripting.

e. An index-name is not defined elsewhere; its appearance in an INDEXED BY clause is its only definition. There may be no items of the same name defined elsewhere. The usage of each index-name is assumed to be INDEX.

f. Subscripting is described in the introduction to this chapter.

PICTURE

Function

The PICTURE clause describes the general characteristics and editing requirements of an elementary item.

General Format

```
[ {PICTURE} IS picture-string ]
  {PIC    }
```

Technical Notes

a. A PICTURE clause may be used only at the elementary level. It may not be used with an item described as USAGE INDEX or COMP-1.

b. A picture-string consists of certain allowable combinations of characters in the COBOL character set used as symbols. These symbols are as follows:

(1) Symbols representing data characters

9 represents a numeric character (0 through 9)

A represents an alphabetic character (A through Z, and the space)

X represents an alphanumeric character (any allowable character)

(2) Symbols representing arithmetic signs and assumed decimal point positioning

V represents the position of the assumed decimal point

P represents an assumed decimal point scaling position

S represents the presence of an arithmetic sign

(3) Symbols representing zero suppression operations

Z represents standard zero suppression (replacement of leading zeroes by spaces)

* represents check protection (replacement of leading zeroes by asterisks)

(4) Symbols representing insertion characters

$ represents a dollar sign (this sign floats from left to right and replaces rightmost leading zero when more than one $ appears)†

†If the CURRENCY SIGN IS clause appears in the SPECIAL-NAMES paragraph, the symbol specified by the literal must be used in all instances in place of the $.

, represents an insertion comma †

. represents an actual decimal point †

B represents an insertion blank

0 represents an insertion zero

(5) Symbols representing editing sign-control symbols

+ represents an editing plus sign — These float and replace rightmost leading zero

- represents an editing minus sign — when more than one + or - appear

CR represents an editing Credit symbol

DB represents an editing Debit symbol

(6) Consecutive repetitions of a picture-string symbol can be abbreviated to the symbol followed by (n), where n indicates the number of occurrences.

c. A maximum number of 30 symbols can appear in a picture-string. Note that the number of symbols in a picture-string and the size of the item represented are not necessarily the same. There are two reasons for this discrepancy. First, the abbreviated form for indicating consecutive repetitions of a symbol may result in fewer symbols in the picture-string than character positions in the item being described. For example, a data item having 40 alphanumeric character positions can be described by a picture-string of only 5 symbols:

PICTURE IS X(40)

The second reason is that some symbols are not counted when calculating the size of the data item being described. These symbols include the V (assumed decimal point), P (decimal point scaling position), and S (arithmetic sign); these symbols do not represent actual physical character positions within the data item. For example, the character-string

S999V99

represents a 5-position data item.

Other size restrictions for numeric and numeric edited items are given under the appropriate headings below.

d. There are five categories of data that can be described with a PICTURE clause: alphabetic, numeric, alphanumeric, alphanumeric edited, and numeric edited. A description of each category is given in the notes below.

e. Definition of an Alphabetic Item

(1) Its picture-string may contain only the symbol A.

(2) It may contain only the 26 letters of the alphabet and the space.

f. Definition of a Numeric Item

(1) Its picture-string may contain only the symbols 9, P, S, and V. It must contain at least one 9.

(2) It may contain only the digits 0 through 9 and an operational sign.

---

†If the DECIMAL-POINT IS COMMA clause appears in the SPECIAL-NAMES paragraph, the function of the comma and decimal point is reversed.

g. Definition of an Alphanumeric Item

(1) Its picture-string can consist of all Xs, or a combination of the symbols A, X, and 9 (except all 9s or all As). The item is treated as if the character-string contained all Xs.

(2) Its contents can be any combination of characters from the complete character set (see Section 1.2, Chapter 1).

h. Definition of an Alphanumeric Edited Item

(1) Its picture-string can consist of any combination of As, Xs, or 9s (it must contain at least one A or one X), plus at least one of the symbols B or 0.

(2) Its contents can be any combination of characters from the complete character set.

i. Definition of a Numeric Edited Item

(1) Its picture-string must contain at least one of the following editing symbols:

, . * + - Z 0 B CR DB $

It may also contain the symbols 9, V, or P.

The allowable sequences are determined by certain editing rules for each symbol and can be found in Note j.

The picture-string must have from 1 to 18 digit positions.

(2) The contents can be any combination of the digits 0 through 9 and the editing characters.

j. The symbols used to define the category of an elementary item and their functions are as follows.

A Each A in the picture-string represents a character position which can contain only a letter of the alphabet or a space.

B Each B in the picture-string represents a character position into which a space character will be inserted during editing.

Examples: (A-FLD contains the value 092469)

	B-FLD picture-string	Result
MOVE A-FLD TO B-FLD.	99B99B99	0 9 Δ 2 4 Δ 6 9
MOVE A-FLD TO B-FLD.	9999BBBB	0 9 2 4 Δ Δ Δ Δ

Also see Note n, "Simple Insertion Editing".

P Each P in the picture-string indicates an assumed decimal point scaling position and is used to specify the location of an assumed decimal point when the point is outside the positions defined for the item. Ps are not counted in the size of the data item. They are counted in determining the maximum number of digit positions (18) allowed in numeric edited items or numeric items. Ps can appear only to the left or right of the picture-string and must appear together. The assumed decimal point is assumed to be to the left of the string of Ps if the Ps are at the left end of the picture-string and to

the right of the string of Ps if the Ps are at the right end of the picture-string. If the V symbol is used in this case, it must appear in either of those positions; it is redundant.

Examples:

PPP9999 (or VPPP9999) defines a data item of four character positions whose contents will be treated as .000nnnn during any decimal point alignment operation (such as in a MOVE or ADD).

9PPP (or 9PPPV) defines a data item of one character position whose contents will be treated as n000 during any decimal point alignment operation.

S An S in a picture-string indicates that the item has an operational sign and will retain the sign of any data stored in it. The S must be written as the leftmost character in the picture-string. If S is not included, all data will be stored in the item as an absolute value and will be treated as positive in all operations. The S symbol is not counted in the size of the data item.

V A V in a picture-string indicates the location of the assumed decimal point and may appear only once in a picture-string. The V does not represent a physical character position and is not counted in the size of the data item. If the assumed decimal point position is at the right of the rightmost character position of the item, the V is redundant (that is, 9999 is functionally equivalent to 9999V).

X Each X in a picture-string represents a character position which can contain any allowable character from the complete character set.

Z Each Z in a picture-string represents the leftmost leading numeric character positions in which leading zeroes are to be replaced by spaces. Each Z is counted in the size of the item.

* Each * in a picture-string represents the leftmost leading numeric character positions in which leading zeros are to be replaced by *. Each * is counted in the size of the item.

Examples: (A-FLD contains the value 00305)

	B-FLD picture-string	Result
MOVE A-FLD TO B-FLD	999999	0 0 0 3 0 5
MOVE A-FLD TO B-FLD	ZZ9999	Δ Δ 0 3 0 5
MOVE A-FLD TO B-FLD	ZZZZZZ	Δ Δ Δ 3 0 5
MOVE A-FLD TO B-FLD	ZZZZ.ZZ	Δ 3 0 5 . 0 0

Also see Note S, "Zero Suppression Editing".

9 Each 9 in a picture-string represents a character position which can contain a digit. Each 9 is counted in the size of the item.

0 Each 0 in a picture-string represents a character position into which a zero will be inserted. It is counted in the size of the item. The 0 symbol works in the same manner as the B symbol.

Each, in a picture-string represents a character position into which a comma will be inserted.

Examples: (A-FLD contains 362577)

	B-FLD picture-string	Result
MOVE A-FLD TO B-FLD	9,999,999	0 , 3 6 2 , 5 7 7
MOVE A-FLD TO B-FLD	Z,ZZZ,ZZZ	Δ Δ 3 6 2 , 5 7 7

Also see Note n, "Simple Insertion Editing".

A . (dot or period) in a picture-string is an editing symbol that represents an actual decimal point. It is used for decimal point alignment and also indicates where a point (.) is to be inserted. This cymbol is counted in the size of the item. Only one . may appear in a picture-string.

Examples: (A-FLD contains 352699)†

	B-FLD picture-string	Result
MOVE A-FLD TO B-FLD	99,999.99	0 3 , 5 2 6 . 9 9
MOVE A-FLD TO B-FLD	ZZ,ZZZ.ZZ	Δ 3 , 5 2 6 . 9 9
MOVE A-FLD TO B-FLD	99999.9999	0 3 5 2 6 . 9 9 0 0

See Noted under MOVE in Chapter 6 for a clarification of the rule governing the third example.

Also see Note O, "Special Editing".

+
-
CR
DB

Each of these symbols is used as an editing sign-control symbol. When used, they represent the character position(s) into which the editing sign-control symbol will be placed. Only one of these symbols can appear in a character-string.

The + and - symbols can appear either at the beginning or at the end of a picture-string. The CR and DB symbols can appear only at the end of a picture-string.

+ The character position containing this symbol will contain a + if the sending field either was unsigned (absolute) or had a positive operational sign; it will contain a - if the sending field had a negative operational sign.

- The character position containing this symbol will contain a space if the sending field either was unsigned (absolute) or had a positive operational sign; it will contain a - if the sending field had a negative operational sign.

CR
DB

Each of these symbols requires two character positions. The character positions containing either of these symbols will contain spaces if the sending field either was unsigned (absolute) or had a positive operational sign; they will contain the symbol specified if the sending field had a negative operational sign.

---

†The caret (^) symbol is used to indicate the location of the assumed decimal point.

Examples: (A-FLD contains 345625, B-FLD contains -345625)[†]

	C FLD picture-string	Result
MOVE A-FLD TO C-FLD	9999.99BCR	3 4 5 6 . 2 5 Δ Δ Δ
MOVE B-FLD TO C-FLD	9999.99BCR	3 4 5 6 . 2 5 Δ C R
MOVE A-FLD TO C-FLD	+9999.99	+ 3 4 5 6 . 2 5
MOVE B-FLD TO C-FLD	+9999.99	- 3 4 5 6 . 2 5
MOVE A-FLD TO C-FLD	-9999.99	Δ 3 4 5 6 . 2 5
MOVE B-FLD TO C-FLD	-9999.99	- 3 4 5 6 . 2 5
MOVE A-FLD TO C-FLD	9999.99DB	3 4 5 6 . 2 5 Δ Δ
MOVE B-FLD TO C-FLD	9999.99DB	3 4 5 6 . 2 5 D B
MOVE B-FLD TO C-FLD	$9999.99+	$ 3 4 5 6 2 5 -

Also see Note p, "Fixed Insertion Editing".

The + and - can also be used to perform floating insertion editing, a combination of zero suppression and symbol insertion. Floating insertion editing is indicated by the occurrence of two or more consecutive + or - symbols at the beginning of the picture-string. The total number of significant positions in the editing field must be at least one greater than the number of significant digits in the data to be edited. The floating + or - moves from left to right through any high-order zeros until a decimal point or the picture character 9 is encountered.

Examples: (A-FLD contains 005625; B-FLD contains -005625)

	C-FLD picture-string	Result
MOVE A-FLD TO C-FLD	++999.99	Δ + 0 5 6 . 2 5
MOVE B-FLD TO C-FLD	++++9.99	Δ Δ - 5 6 . 2 5
MOVE ZERO TO C-FLD	++999.99	Δ + 0 0 0 . 0 0
MOVE ZERO TO C-FLD	+++++.++	Δ Δ Δ Δ Δ Δ Δ Δ

(In order for floating to go past decimal point, all numeric positions of item must be represented by the floating insertion symbol)

MOVE A-FLD TO C-FLD	--999.99	Δ Δ 0 5 6 . 2 5
MOVE B-FLD TO C-FLD	--999.99	Δ - 0 5 6 . 2 5
MOVE ZERO TO C-FLD	---99.99	Δ Δ Δ 0 0 . 0 0
MOVE ZERO TO C-FLD	-----.--	Δ Δ Δ Δ Δ Δ Δ Δ

Also see Note o, "Floating Insertion Editing".

[†]The caret (^) symbol is used to indicate the location of the assumed decimal point.

Note that the + and - symbols are distinct from the S (operational sign) symbol. Normally, the + and - symbols are used to describe display items that are to appear on some printed report; they provide visual sign indication and cannot be used with items appearing as operands in arithmetic statements.

$ A $ (or the symbol specified by the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph) represents the character position into which a $ (or the currency symbol) is to be placed. This symbol is counted in the size of the item.

Examples: (A-FLD contains 345675)
^

	B-FLD character-string	Result
MOVE A-FLD TO B-FLD	$9,999.99	$3,456.75
MOVE A-FLD TO B-FLD	$999,999.99	$003,456.75

Also see Note p, "Fixed Insertion Editing".

The $ symbol can also be used to perform floating insertion editing. Floating insertion editing is indicated by the occurrence of two or more consecutive $ symbols at the beginning of the character string. The total number of significant positions in the editing field must be at least one greater than the number of significant digits in the data to be edited. The floating $ symbol floats from left to right through any high-order zeros until a decimal point or the picture character 9 is encountered.

Examples: (A-FLD contains 005625)
^

	B-FLD picture-string	Result
MOVE A-FLD TO B-FLD	$$9,999.99	Δ$0,056.25
MOVE A-FLD TO B-FLD	$$$,$$$.99	ΔΔΔΔ$56.25
MOVE ZERO TO B-FLD	$$$,999.99	ΔΔΔ$000.00
MOVE ZERO TO B-FLD	$$$,$$$.$$	ΔΔΔΔΔΔΔΔΔΔ

Also see Note q, "Floating Insertion Editing".

k. There are two general methods of performing editing in the PICTURE clause:

(1) insertion, or

(2) suppression and replacement.

There are four types of insertion editing available:

(1) Simple insertion

(2) Special insertion

(3) Fixed insertion

(4) Floating insertion

There are two types of suppression and replacement editing:

(1) Zero suppression and replacement with spaces

(2) Zero suppression and replacement with asterisks

l. The type of editing that may be performed upon an item depends on the category to which the item belongs.

Category	Type of Editing Allowed
Alphabetic	None
Numeric	None
Alphanumeric	None
Alphanumeric edited	Simple insertion: 0 and B
Numeric Edited	All (except for the restrictions given in Note m)

m. Floating insertion editing and zero suppression/replacement editing are mutually exclusive in a PICTURE clause. Only one type of replacement can be used with zero suppression in a PICTURE clause.

n. Simple Insertion Editing (, B 0)

The , (comma), B (space), and 0 (zero) constitute those editing symbols used in simple insertion editing. These insertion characters represent the character position in the item into which the character will be inserted. These symbols are counted in the size of the item.

o. Special Insertion Editing (.)

The . (decimal point) symbol is used in special insertion editing. In addition to its use as an insertion character, it also represents the position of the decimal point for decimal point alignment. This symbol is counted in the size of the item. The symbols . and V (assumed decimal point) are mutually exclusive in a PICTURE clause. If the . is the last symbol in the character-string, it must be immediately followed by one of the punctuation characters (semi-colon or period) followed by a space.

p. Fixed Insertion Editing ($ + - CR DB)

The currency symbol ($) and the editing sign control characters (+ - CR DB) constitute the characters used in fixed insertion editing. Only one $ and one of the editing sign control characters can be used in a PICTURE character-string. When the symbols CR or DB are used, they represent two character positions in determining the size of the item. The symbols + or - when used must be the leftmost or rightmost character positions to be counted in the size of the item. The $ when used must be the leftmost character position to be counted in the size of the item, except that it can be preceded by a + or - symbol. A fixed insertion editing character appears in the same character position in the edited item as it occupied in the PICTURE character-string.

Editing sign control symbols produce the following results depending on the value of the data being edited.

Editing Symbol in PICTURE character-string	Result	
	Data Positive	Data Negative
+	+	-
-	space	-
CR	2 spaces	CR
DB	2 spaces	DB

q. Floating Insertion Editing ($$ ++ --)

The $ and the editing sign control symbols + and - are the floating insertion editing characters and are mutually exclusive in a given PICTURE string.

Floating insertion editing is indicated in a PICTURE character-string by using a string of at least two of the allowable insertion characters to represent the leftmost numeric character positions into which the insertion characters can be floated. Any of the simple insertion characters embedded in the string of floating insertion characters or to the immediate right of this string are part of the floating string.

In a PICTURE character-string, there are only two ways of representing floating insertion editing:

(1) Represent any two or more of the leading numeric character positions on the left of the decimal point by the insertion character. The result is that a single insertion character will be placed in the character position immediately preceding the leftmost nonzero digit of the data being edited or in the character position immediately preceding the decimal point, or in the character position represented by the rightmost insertion character, whichever is encountered first.

(2) Represent all numeric character positions in the character-string by the insertion character. If the value is not zero, the result is the same as when the insertion character appears only to the left of the decimal point. If the value is zero, the entire item is set two spaces.

A picture-string containing floating inserting characters must contain at least one more floating insertion character than the maximum number of significant digits in the item to be edited. For example, a data field containing five significant digit positions requires an editing field of at least six significant positions.

All floating insertion characters are counted in the size of the item.

r. Zero Suppression Editing (Z *)

The suppression of leading zeroes and commas in a data field is indicated by the use of the Z or the * symbol in a picture-string. These symbols are mutually exclusive in a given picture-string. Each suppression symbol is counted in the size of the item. If a Z is used, the replacement character is a space. If an * is used, the replacement character is an *.

Zero suppression and replacement is indicated by a string of one or more Zs or *s to represent the leading numeric-character positions which are to be replaced when the associated character position in the data contains a leading zero. Any of the simple insertion characters embedded in this string of zero suppression symbols or to the immediate right of this string are part of the string.

If the zero suppression symbols appear only to the left of the decimal point, any leading zero in the data that corresponds to a zero suppression symbol in the string is replaced by the replacement character.

Suppression terminates at the first nonzero digit in the data represented by the suppression symbol in the string or at the decimal point, whichever is encountered first.

If all numeric character positions in the picture-string are represented by the suppression symbol and the value of the data is not zero, the result is the same as if the suppression characters were only to the left of the decimal point. If the value is zero, the entire item will be set to the replacement character (with the exception of the decimal point if the suppression symbol is an *).

When the * is used and the clause BLANK WHEN ZERO appears in the same entry and zeros are moved to the field, all character positions with the exception of the decimal point are replaced by *.

s. The symbols + - * Z and $ when used as floating replacement characters are mutually exclusive within a given picture-string.

t. The following chart shows the order of precedence of the various picture-string symbols. Each "Y" on the chart indicates that the symbol in the top row directly above can precede the symbol at the left of the row in which the "Y" appears.

{ } indicate that the symbols are mutually exclusive.

The P and the fixed insertion + and - appear twice.

P9, +9, and -9 represent the case where these symbols appear to the left of any numeric positions in the string.

9P, 9+, and 9- represent the case where these symbols appear to the right of any numeric positions in the string.

The Z, *, and the floating ++, --, and $$ also appear twice.

Z., *., $$., ++., and --. represent the case where these symbols appear before the decimal point position.

.Z, .*, .$$, .++, and .-- represent the case where these symbols appear following the decimal point position.

		FIXED INSERTION								OTHER											
		B	0	,	.	{+9 -9}	{9+ 9-}	{CR DB}	$	A X	P9	9P	S	V	{Z. *.}	{.Z .*}	9	{++. --.}	{.++ .--}	$$.	.$$
FIXED INSERTION	B	Y	Y	Y	Y	Y			Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y
	0	Y	Y	Y	Y	Y			Y	Y	Y			Y	Y	Y	Y	Y	Y	Y	Y
		Y	Y	Y	Y	Y			Y		Y			Y	Y	Y	Y	Y	Y	Y	Y
		Y	Y	Y		Y			Y		Y				Y		Y	Y		Y	
	{+9 -9}										Y			Y							
	{9+ 9-}	Y	Y	Y	Y				Y		Y			Y	Y	Y	Y			Y	Y
	{CR DB}	Y	Y	Y	Y				Y		Y			Y	Y	Y	Y			Y	Y
	$					Y					Y			Y							
OTHER	A X	Y	Y							Y							Y				
	P9										Y		Y	Y							
	9P	Y	Y	Y		Y	Y	Y	Y			Y	Y		Y		Y	Y		Y	
	S																				
	V	Y	Y	Y		Y	Y	Y	Y			Y	Y		Y		Y	Y		Y	
	{Z. *.}	Y	Y	Y		Y			Y						Y						
	{.Z .*}	Y	Y	Y	Y	Y			Y		Y			Y	Y	Y					
	9	Y	Y	Y	Y	Y			Y	Y	Y		Y	Y	Y		Y	Y		Y	
	{++. --.}	Y	Y	Y					Y									Y			
	{.++ .--}	Y	Y	Y	Y				Y		Y			Y				Y	Y		
	$$.	Y	Y	Y		Y														Y	
	.$$	Y	Y	Y	Y	Y					Y			Y						Y	Y

REDEFINES

Function

The REDEFINES clause allows the same core memory area to be allocated to two or more data items.

General Format

level-number data-name-1 REDEFINES data-name-2

Technical Notes

a. The REDEFINES clause, when used, must immediately follow data-name-1.

b. The level-numbers of data-name-1 and data-name-2 must be identical.

c. This clause must not be used for level-number 66 or 88 items. Also, it must not be used for level-01 entries in the FILE SECTION; implicit redefinition is provided by specifying more than one data-name in the DATA RECORDS ARE clause in the FD.

d. When the level-number of data-name-2 is other than level-01, it must specify a storage area of the same size as data-name-1. FILLER items may be used to comply with this rule.

e. This entry must immediately follow the entries describing the area being redefines.

f. The data description entry for data-name-2 cannot contain an OCCURS clause or be subordinate to an entry that contains an OCCURS clause. Also, the redefinition entries cannot contain VALUE clauses, except in condition-name (level-88) entries.

g. Data-name-2 must not be qualified.

h. The following example illustrates the use of the REDEFINES entry. The entries shown cause AREA-A and AREA-B to occupy the same area in memory.

```
03 AREA-A.
    04 FIELD-1 PICTURE IS X(7).
    04 FIELD-2 PICTURE IS A(13).
    04 FIELD-3.
```

```
        05  SUBFIELD-1  PICTURE IS
            S999V99 USAGE IS COMP.
        05  SUBFIELD-2  PICTURE IS
            S999V99 USAGE IS COMP.
    03  AREA-B REDEFINES AREA-A.
        04  FIELD-A  PICTURE IS X(22).
        04  FIELD-B  PICTURE IS X(5).
        04  FILLER PICTURE IS X(5).
```

Note how the length of each area is calculated so that AREA-B can be defined so that its size is equal to that of AREA-A.

AREA-A:	FIELD-1	7	6-bit characters	(DISPLAY-6 assumed)
	FIELD-2	13	6-bit characters	(DISPLAY-6 assumed)
	SUBFIELD-1	6	6-bit characters	(COMP items occupy one word, or six 6-bit character positions)
	SUBFIELD-2	6	6-bit characters	(COMP items occupy one word, or six 6-bit character positions)
Total 6-bit characters		32		
AREA-B:	FIELD-A	22	6-bit characters	(DISPLAY-6 assumed)
	FIELD-B	5	6-bit characters	(DISPLAY-6 assumed)
	FILLER	5	6-bit characters	(needed to make AREA-B size equal to AREA-A)
Total 6-bit characters		32		

RENAMES (level-66)

Function

The RENAMES clause permits alternate, possibly overlapping, groupings of elementary items.

General Format

66 data-name-1 RENAMES data-name-2 [THRU data-name-3] .

Technical Notes

a. All RENAMES entries associated with items in a given record must immediately follow the last data description entry for that record.

```
01  data-name-a
      .
    (data description entries)
      .
      .
    (level-66 entries associated with this logical record)
01  data-name-b.
      .
      .
```

b. Data-name-1 cannot be used as a qualifier, and can be qualified only by the names of the level-01 or FD entries associated with it.

c. Data-name-2 and data-name-3 must be the names of items in the associated logical record and cannot be the same data-name.

Neither data-name-2 nor data-name-3 can have a level-number of 01, 66, 77, or 88. Neither of these data-names can have an OCCURS clause in its data description entry, nor be subordinate to an item that has an OCCURS clause in its data description entry.

Data-name-2 must precede data-name-3 in the record description, and data-name-3 cannot be subordinate to data-name-2. If there is any associated redefinition (REDEFINES), the ending point of data-name-3 must logically follow the beginning point of data-name-2.

When data-name-3 is specified, data-name-1 is a group item that includes all elementary items starting with data-name-2 (if data-name-2 is an elementary item) or the first elementary item in data-name-2 (if data-name-2 is a group item) and concluding with data-name-3 (or the last elementary item in data-name-3).

If data-name-3 is not specified, data-name-2 can be either a group or elementary item. If it is a group item, data-name-1 is treated as a group item and includes all elementary items in data-name-2; if data-name-2 is an elementary item, data-name-1 is treated as an elementary item with the same descriptive clauses.

d. The following examples illustrate the use of the RENAMES entry.

```
01  RECORD-NAME.
    02  FIRST-PART.
        03  PART-A.
            04  FIELD-1 PICTURE IS ...
            04  FIELD-2 PICTURE IS ...
            04  FIELD-3 PICTURE IS ...
        03  PART-B.
            04  FIELD-4 PICTURE IS ...
            04  FIELD-5.
                05  FIELD-5A PICTURE IS ...
                05  FIELD-5B PICTURE IS ...
    02  SECOND-PART.
        03  PART-C.
            04  FIELD-6 PICTURE IS ...
            04  FIELD-7 PICTURE IS ...
    66  SUBPART RENAMES PART-B THRU PART-C.
    66  SUBPART1 RENAMES FIELD-3 THRU SECOND-PART.
    66  SUBPART2 RENAMES FIELD-5B THRU FIELD-7.
    66  AMOUNT RENAMES FIELD-7.
```

SYNCHRONIZED

Function

The SYNCHRONIZED clause specifies the positioning of an elementary item within a computer word (or words).

General Format

Technical Notes

a. This clause can appear only in the data description of an elementary item.

b. This clause specifies that the item being defined is to be placed in an integral number of computer words and that it is to begin or end at a computer word boundary. No other adjacent fields are to occupy these words. The unused positions, however, must be counted when calculating (1) the size of any group to which this elementary item belongs, and (2) the computer core allocation when the item appears as the object of a REDEFINES clause. However, when a SYNCHRONIZED item is referenced, the original size of the item (as indicated by the PICTURE clause) is used in determining such things as truncation, justification, and overflow.

c. SYNCHRONIZED LEFT or SYNC LEFT specifies that the item is to be positioned in such a way that it will begin at the left boundary of a computer word.

SYNCHRONIZED RIGHT or SYNC RIGHT specifies that the item is to be positioned in such a way that it will terminate at the right boundary of a computer word.

d. When the SYNCHRONIZED clause is specified for an item within the scope of an OCCURS clause, each occurrence of the item is SYNCHRONIZED.

e. Any FILLER required to position the item as specified will be automatically generated by the compiler. The content of this FILLER is indeterminate.

f. COMP(UTATIONAL) and COMP(UTATIONAL)-1 items are always implicitly SYNCHRONIZED RIGHT.

USAGE

Function

The USAGE clause specifies the format of a data item in computer storage.

General Format

```
[ USAGE IS { COMPUTATIONAL      } ]
           { COMP               }
           { COMPUTATIONAL-1    }
           { COMP-1             }
           { DISPLAY            }
           { DISPLAY-6          }
           { DISPLAY-7          }
           { INDEX              }
```

Technical Notes

a. The USAGE clause can be written at any level. If it is written at a group level, it applies to each elementary item in the group. The USAGE clause of an elementary item cannot contradict the USAGE clause of a group to which the item belongs.

The implied USAGE of a group item is DISPLAY-7 if all items subordinate to it are defined as DISPLAY-7; otherwise, its USAGE is DISPLAY-6 (DISPLAY).

b. This clause specifies the manner in which a data item is represented within computer memory.

c. COMPUTATIONAL (COMP)

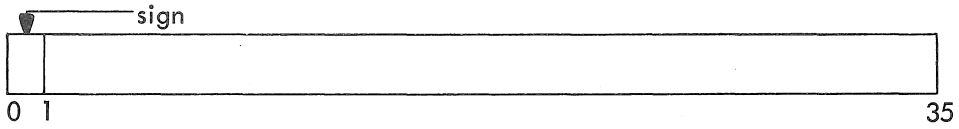
(1) COMP is equivalent to COMPUTATIONAL.

(2) A COMPUTATIONAL item represents a value to be used in computations and must be numeric. Its picture-string can contain only the symbols: 9 S V P
Its value is represented as a binary number with an assumed decimal point.

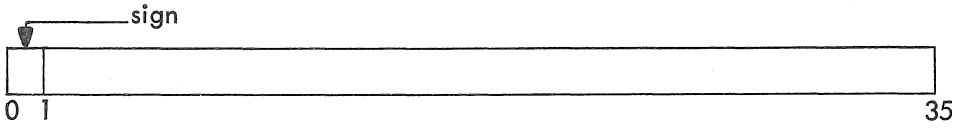
(3) If a group item is described as COMPUTATIONAL, the elementary items in the group are COMPUTATIONAL. However, the group item itself is not COMPUTATIONAL and cannot be used as an operand in arithmetic computations.

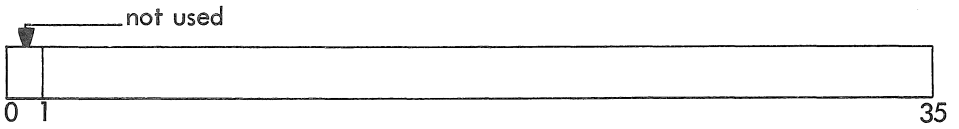
(4) COMPUTATIONAL items of not more than 10 decimal positions will be SYNCHRONIZED RIGHT in one computer word. COMPUTATIONAL items of more than 10 decimal positions will be SYNCHRONIZED RIGHT in two full computer words.

(5) The following illustrations give the format of a COMPUTATIONAL item.



1-WORD COMPUTATIONAL ITEM





2-WORD COMPUTATIONAL ITEM

d. COMPUTATIONAL-1 (COMP-1)

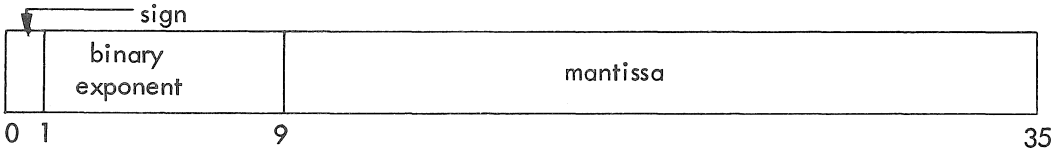
(1) COMP-1 is equivalent to COMPUTATIONAL-1.

(2) A COMPUTATIONAL-1 item can contain a value, in floating point format, to be used in computations and must be numeric. A COMP-1 item must not have a PICTURE.

(3) If a group item is described as COMPUTATIONAL-1, the elementary items within the group are COMPUTATIONAL-1. However, the group item itself is not COMPUTATIONAL-1 and cannot be used as an operand in arithmetic computations.

(4) COMPUTATIONAL-1 items will be SYNCHRONIZED in one full computer word.

(5) The following illustration gives the format of a COMPUTATIONAL-1 item.



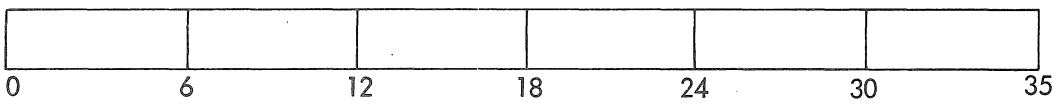
e. DISPLAY-6 (DISPLAY)

(1) DISPLAY is equivalent to DISPLAY-6.

(2) A DISPLAY-6 item represents a string of 6-bit characters. Its picture-string may contain any picture symbols.

(3) DISPLAY-6 items may be SYNCHRONIZED LEFT or SYNCHRONIZED RIGHT, as desires; otherwise, they may share a computer word with other DISPLAY-6 items.

(4) The illustration below gives the format of a DISPLAY-6 word.



(5) If the USAGE clause is omitted for an elementary item, its USAGE is assumed to be DISPLAY-6 (DISPLAY).

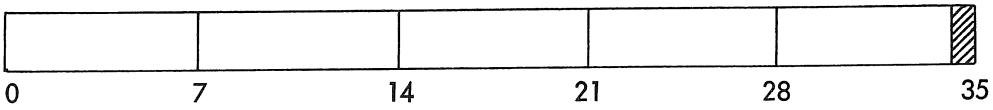
f. DISPLAY-7

(1) A DISPLAY-7 item represents a string of 7-bit ASCII characters. Its picture-string may contain any picture symbols.

(2) If any item in a record is DISPLAY-7, all items in that record must be DISPLAY-7.

(3) DISPLAY-7 items can be SYNCHRONIZED LEFT or SYNCHRONIZED RIGHT, as desired; otherwise, they may share a computer word with other items. If the item is SYNCHRONIZED RIGHT, the last character of the item will end in bit 34 of a computer word.

(4) The illustration below gives the format of a DISPLAY-7 word.



g. INDEX

(1) An elementary item described as USAGE INDEX is called an index data-item. It is treated as a COMP item with PICTURE S9(5).

(2) An index data-item must not have a PICTURE.

(3) If a group item is described as INDEX, the elementary items within the group are treated as INDEX. However, the group item itself is not INDEX and cannot be used as an operand in arithmetic statements.

(4) Index data items and index-names (defined in the OCCURS clause by the INDEXED BY option) are equivalent.

VALUE

Function

The VALUE clause defines the initial value of WORKING-STORAGE items, and the values associated with condition-names (level-88).

General Format

FORMAT 1:
```
[ VALUE IS literal ]
```

FORMAT 2:
```
[ { VALUE IS   }                                              ]
[ { VALUES ARE } literal-1 [THRU literal-2]                   ]
[                [, literal-3 [THRU literal-4]] ... ]
```

Technical Notes

a. Format 2 can be specified only for level-88 items.

b. In the FILE SECTION, the VALUE clause can be used only with level-88 items. In the WORKING-STORAGE SECTION, it can be used at all levels, except level-66. It must not be stated in a data description entry that contains an OCCURS clause or that is subordinate to an entry containing an OCCURS clause. Also, it must not be stated in an entry that contains a REDEFINES clause or that is subordinate to an entry that contains a REDEFINES clause (unless the VALUE clause is part of a level-88 entry).

c. If the VALUE clause is used at a group level, the literal must be a figurative constant or a nonnumeric literal. The group item is initialized to this value without consideration for the individual elementary or group items contained within this group. No VALUE clauses can appear at subordinate levels within the group.

d. If no VALUE clause appears for a WORKING-STORAGE item, the initial value of the item is unpredictable.

e. More information concerning Format 2 can be found under "condition-name (level-88)" in this chapter.

f. The VALUE clause must not conflict with other clauses in the data description entry or in the data description entries within the hierarchy of the item. The following rules apply:

(1) If the category of an item is numeric, all literals in the VALUE clause must be numeric. All literals in a VALUE clause must have a value within the range of values indicated by the PICTURE clause; for example, an item with PICTURE PPP9 may have only the values in the range .0000 through .0009.

(2) If the category of the item is alphabetic or alphanumeric, all literals in the VALUE clause must be nonnumeric literals. The literal will be aligned according to the normal alignment rules (see "JUSTIFIED") except that the number of characters in the literal must not exceed the size of the item.

(3) If the category of an item is numeric-edited or alphanumeric-edited, no editing of the value is performed.

g. The figurative constants SPACE(S), ZERO(E)(S), QUOTE(S), LOW-VALUE(S), and HIGH-VALUE(S) may be substituted for a literal. If the item is numeric, only ZERO(E)(S) is allowed.

Chapter 6
The PROCEDURE DIVISION

This processing is described by a series of COBOL procedure statements. Statements, sentences, paragraphs, and sections are generally described in Section 1.7 in Chapter 1. The PROCEDURE DIVISION must contain at least one paragraph, and each paragraph must contain at least one sentence. Sections are optional and permit a group of consecutive paragraphs to be referenced by a single procedure-name; sections can also be used for segmentation purposes (see "Segmentation"). If any section appears in the PROCEDURE DIVISION, then all paragraphs must appear within a section.

The first entry in the PROCEDURE DIVISION of a source program must be the division-header.

```
PROCEDURE DIVISION.
```

The next entry must be either the DECLARATIVES header (see "USE"), or a paragraph-name or section-name.

6.1 SYNTACTIC FORMAT OF THE PROCEDURE DIVISION

The PROCEDURE DIVISION consists of a series of procedure statements grouped into sentences, paragraphs, and sections. By grouping the statements in this manner, reference can be made to them via a procedure-name (i.e., a paragraph-name or a section-name). The order in which procedure-statements are executed can be controlled by using the sequence-control verbs ALTER, GO TO, and PERFORM.

6.1.1 Statements and Sentences

Statements and sentences are generally described in Chapter 1. Statements fall into three categories: imperative, conditional, and compiler-directing, depending upon the verb used. Verbs, in turn, are also classified into certain categories. These categories and their relationship to the three statement categories are given in Table 6-1.

Table 6-1
Procedure Verb and Statement Categories

Verb	Verb Category	Statement Category
ADD COMPUTE DIVIDE MULTIPLY SUBTRACT	ARITHMETIC	IMPERATIVE
ALTER GO TO PERFORM	SEQUENCE-CONTROL	
EXAMINE MOVE SET	DATA MOVEMENT	
ENTER SORT STOP	MISCELLANEOUS	
ACCEPT CLOSE DISPLAY OPEN READ SEEK WRITE	I-O	
IF	CONDITIONAL	CONDITIONAL
COPY EXIT NOTE USE	COMPILER-DIRECTING	COMPILER-DIRECTING

A statement or sequence of statements terminated by a period forms a sentence. Sentences are classified into the same three categories as statements.

An imperative sentence consists solely of one or more imperative statements. Except for imperative sentences containing one of the sequence control verbs, control passes to the next procedural sentence following execution of the imperative sentence. If a GO TO or STOP RUN statement is present in an imperative sentence, it must be the last statement in the sentence.

A conditional sentence performs some test and, on the basis of the results of that test, determines whether a "true" or a "false" path should be taken. A conditional sentence is one that contains the conditional verb (IF) or one of the option clauses ON SIZE ERROR (used with arithmetic verbs), AT END (used with the READ verb), or INVALID KEY (used with the READ verb for mass storage devices).

A compiler-directing sentence consists of a single compiler-directing statement. Compiler-directing sentences are used to indicate the end point of a PERFORM loop (EXIT), insert comments in the PROCEDURE DIVISION (NOTE), and specify procedures for input-output errors and label handling (USE). Generally, compiler-directing sentences generate no object program coding.

6.1.2 Paragraphs

A single sentence or a group of sequential sentences can be assigned a paragraph-name for reference. The paragraph-name must begin in Area A (see Chapter 2) and terminate with a period. The first sentence of the paragraph can begin after the space following this period or it can begin on the next line, beginning in Area B.

A paragraph-name must be unique within its section, but need not be unique within the program. A non-unique paragraph-name must be qualified by its section-name except when it is references from within its own section.

6.1.3 Sections

A single paragraph or a group of sequential paragraphs can be assigned a section-name for reference. The section-name must begin in Area A and be followed by the word SECTION followed by a priority number, if desired, followed by a terminating period.

```
section-name SECTION nn.
```

If the section-name is in the DECLARATIVES portion, it may not have a priority number. A USE statement may appear following the terminating space after the period.

The section-name applies to all paragraphs following it until another section-header is encountered.

All section-names must be unique within a program. Sections are optional within the PROCEDURE DIVISION, but if a DECLARATIVES portion is used there must be a named section immediately following the END DECLARATIVES statement.

When a section-name is referenced, the word SECTION is not allowed in the reference.

6.2 SEQUENCE OF EXECUTION

In the absence of sequence-control verbs, sentences are executed consecutively within paragraphs, paragraphs are executed consecutively within sections, and sections are executed consecutively within the PROCEDURE DIVISION (with the exception of sections within the DECLARATIVES portion, which are executed individually when the related condition occurs).

6.3 SEGMENTATION AND SECTION-NAME PRIORITY NUMBERS

COBOL source programs can be written to enable certain portions of the PROCEDURE DIVISION code to share the same core memory area at object run time, thus decreasing the amount of core required to run the object program. The method used to achieve this reduction is called segmentation.

Segmentation consists of dividing the PROCEDURE DIVISION sections into logically related groupings called segments. The programmer defines a segment by assigning the same priority-number (a priority-number follows the word SECTION in the section-header, and can be in the range 00 through 99) to all the sections he wants included in that segment; these sections need not appear consecutively in the source program.

Segments are classified into three groups, depending upon their priority-number. These three groups are described in Table 6-2.

Table 6-2
Types of Segments

Priority-Number	Type	Description
None, or 00 up to SEGMENT-LIMIT minus 1	Resident Segment	This segment is always resident in core and is never overlaid.
SEGMENT-LIMIT up to 49	Nonresident; ALTERed GO TOs retained	These segments are nonresident and are brought into core when needed. Any ALTERed GO TOs retain their most recently set values.

Table 6-2 (Cont)
Types of Segments

Priority-Number	Type	Description
50 through 99	Nonresident; ALTERed GO TOs reset	These segments are also nonresident and are brought into core when needed. Any ALTERed GO TOs do not retain their latest values, but are reset to their original setting each time the segment is entered from another segment.

In addition to the resident segment, all data areas described in the DATA DIVISION are resident at all times. Thus, memory can be thought of as being divided into two parts:

a. A resident area, in which reside all data areas and the resident segment, and

b. A nonresident area, equal to the size of the largest nonresident segment, into which each nonresident segment is read when needed. Since each nonresident segment reads into the same memory area, any previous nonresident segment in that area is overlaid and must be brought in again when it is to be executed again.

The resident segment should consist of those sections that constitute the main portion of the processing. Infrequently used sections can be allocated to the nonresident segments.

6.4 ARITHMETIC EXPRESSIONS

An arithmetic expression is an identifier of a numeric elementary item, or a numeric literal, or such identifiers and literals separated by arithmetic operators.

Algebraic negation can be indicated by a unary minus symbol.

6.4.1 Arithmetic Operators

There are five arithmetic operators that may be used in arithmetic expressions. They are represented by specific character symbols that must be preceded by a space and followed by a space.

Arithmetic Operator	Meaning
+	Addition or unary plus
-	Subtraction or unary minus
*	Multiplication
/	Division
**	Exponentiation
↑	Exponentiation

6.4.2 Formation and Evaluation Rules

The following rules for formation and evaluation apply to arithmetic expressions.

a. Parentheses specify the order in which elements within an arithmetic expression are to be evaluated. Expressions within parentheses are evaluated first. Within a nest of parentheses, the evaluation proceeds from the elements within the innermost pair of parentheses to the outermost pair of parentheses. When parentheses are not used, or parenthesized expressions are at the same level of inclusiveness, the following hierarchal order of operations is implied:

First:	unary +, unary -	
then	** and ↑	(exponentiation)
then	* and /	(multiplication and division)
and then	+ and -	(addition and subtraction)

b. When the order of a sequence of operations on the same hierarchal level (e.g., a sequence of + and - operations) is not completely specified by use of parentheses, the order of operations is from left to right.

c. An arithmetic expression may begin only with one of the following:

(- + variable

and may end only with one of the following:

) variable

d. There must be a one-to-one correspondence between left and right parentheses in an arithmetic expression; each left parenthesis must precede its corresponding right parenthesis.

6.5 CONDITIONAL EXPRESSIONS

A conditional expression causes the object program to select between alternate paths (called the true and false paths) of control depending upon the truth value of a test. Conditional expressions can be used in conditional (IF) statements and in PERFORM statements (options 3 and 4). A conditional expression can be one of the following types:

Relation condition	(greater than, equal to, less than)
Class condition	(numeric or alphabetic)
Condition-name condition	(level-88 condition-names)
Switch-status condition	(SPECIAL-NAMES paragraph)
Sign condition	(positive, negative, zero)

Each of these types is discussed below.

6.5.1 Relation Condition

A relation condition causes a comparison of two operands, each of which may be an identifier, a literal, a figurative constant, or an arithmetic expression. Comparison of two numeric operands is permitted regardless of their formats as described by their respective USAGE clauses. Comparison of two operands is permitted if each is either DISPLAY-6 or DISPLAY-7. However, for all other comparisons, the operands must be described as having the same USAGE.

A numeric-edited operand may not be compared to a numeric operand. An alphanumeric operand may not be compared to a numeric operand that has decimal places.

6.5.1.1 Format of a Relation Condition

The general format for a relation condition is

{ identifier-1 / literal-1 / arithmetic-expression-1 / figurative-constant-1 } relational-operator { identifier-2 / literal-2 / arithmetic-expression-2 / figurative-constant-2 }

The first operand is called the subject of the condition; the second operand is called the object of the condition. Either the subject or the object must be an identifier or an arithmetic expression.

6.5.1.2 Relational Operators

Relational operators specify the type of comparison to be made in the relation condition. Relational operators must be preceded by a space and followed by a space.

Relational Operator	Meaning
IS [NOT] GREATER THAN IS [NOT] > THAN	Greater than, not greater than
IS [NOT] LESS THAN IS [NOT] < THAN	Less than, not less than
IS [NOT] EQUAL (EQUALS) TO IS [NOT] = TO	Equal to, not equal to

6.5.1.3 Comparison of Numeric Items

For operands with a numeric category, a comparison results in the determination that the algebraic value of one of the operands is less than, equal to, or greater than the other operand. The number of digits contained in the operands is not significant. Zero is considered a unique value regardless of the sign (i.e., +0 and -0 are considered equal). Unsigned numeric operands are considered positive for purposes of comparison.

6.5.1.4 Comparison of Nonnumeric Items - For operands whose category is nonnumeric (or where one operand is numeric and the other is nonnumeric), a comparison results in the determination that one of the operands is less than, equal to, or greater than the other operand with respect to a specified collating sequence of characters (see Appendix B). The size of an operand is the total number of characters in the operand.

There are two cases to consider: operands of equal size, and operands of unequal size.

a. Operands of equal size - If the operands are of equal size, characters in corresponding character positions of the two operands are compared, starting at the higher-order (leftmost) end and continuing through the low-order end. If all pairs of characters compare equally through the last pair, the operands are considered to be equal. If they do not all compare equally, the first pair of unequal characters encountered is compared to determine their relative position in the collating sequence. The operand containing the character that is positioned higher in the collating sequence is considered to be the greater operand.

b. Operands of unequal size - If the operands are of unequal size, the comparison of characters proceeds from the high-order end to the low-order end until either

(1) A pair of unequal characters is encountered, or

(2) One of the operands has no more characters to compare.

If a pair of unequal characters is encountered, the comparison is determined in the manner described for equal-sized operands.

If the end of one of the operands is encountered before unequal characters are encountered, this shorter operand is considered to be less than the longer operand unless the remaining characters in the longer operand are spaces, in which case the two operands are considered equal.

6.5.2 Class Condition

The class condition determines whether the operand is numeric (i.e., whether it consists entirely of the digits 0 through 9, with or without an operational sign) or alphabetic (i.e., whether it consists entirely of the characters A through Z and the space).

6.5.2.1 Format of a Class Condition - The general format of a class condition is

identifier IS [NOT] { NUMERIC / ALPHABETIC }

The identifier must be described, implicitly or explicitly, as DISPLAY, DISPLAY-6, or DISPLAY-7.

6.5.2.2 The NUMERIC Test - The NUMERIC test cannot be used with an item that is described as alphabetic. If the item being tested is not described as containing an operational sign, it will be considered numeric only if the contents are numeric and an operational sign is not present.

6.5.2.3 The ALPHABETIC Test - The ALPHABETIC test cannot be used with an item that is described as numeric. The item being tested is determined to be alphabetic only if the contents consist entirely of any combination of the alphabetic characters A through Z and the space.

6.5.3 Condition-Name Condition

In a condition-name condition, a conditional variable is tested to determine whether or not its value is equal to one of the values associated with a condition-name (level-88).

6.5.3.1 Format of a Condition-Name Condition - The general format for a condition-name condition is

[NOT] condition-name

If the condition-name is associated with a range of values, then the conditional variable is tested to determine whether or not its value falls within this range, including the end values.

The rules for comparing a conditional variable with a condition-name value are the same as those specified for relation conditions.

The result of the test is true if one of the values associated with the condition-name equals the value of its associated conditional variable.

6.5.4 Switch-Status Condition

A switch-status condition determines the on or off status of a hardware switch.

6.5.4.1 Format of a Switch-Status Condition - The general formats for a switch-status condition are

Format 1: [NOT] condition-name

Format 2: mnemonic-name IS [NOT] { ON / OFF }

Format 3: SWITCH (integer) IS [NOT] { ON / OFF }

In format 1, condition-name is associated with a SWITCH IS ON or OFF STATUS clause in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION.

In format 2, mnemonic-name is associated with a SWITCH (not an ON or OFF STATUS) in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION.

In format 3, integer must be in the range from 0 through 35.

In format 1, the result of the test is true if the switch is [NOT] set to the position associated with the condition-name.

In formats 2 and 3, the result of the test is true if the switch is [NOT] set to the position specified in the condition.

6.5.5 Sign Condition

The sign condition determines whether or not the algebraic value of a numeric operand is less than, greater than, or equal to zero.

6.5.5.1 Format of a Sign Condition - The general format for a sign condition is

$$\left\{\begin{array}{l}\text{identifier}\\ \text{arithmetic-expression}\end{array}\right\} \text{ IS [NOT] } \left\{\begin{array}{l}\underline{\text{POSITIVE}}\\ \underline{\text{NEGATIVE}}\\ \underline{\text{ZERO}}\end{array}\right\}$$

An operand is positive if its value is greater than zero, negative if its value is less than zero, and zero if its value is equal to zero (the sign is ignored if the value is zero).

6.5.6 Logical Operators

The interpretation of any of the above conditions is reversed by preceding the condition with the logical operator NOT. Any of the above types of conditions can be combined by either of two logical operators. A logical operator must be preceded by a space and followed by a space.

Logical Operator	Meaning
OR	Entire condition is true if either or both of the simple conditions are true.
AND	Entire condition is true if both of the simple conditions are true.

6.5.7 Formation and Evaluation Rules

A conditional expression can be composed of either a simple-condition or a compound-condition. A simple-condition is one that performs a single test. A compound-condition is one that contains a string of simple-conditions connected by the logical operators AND, OR. A compound-condition can contain any combination of types of conditional expressions (relational, class, condition-name, switch-status, and sign).

The evaluation rules for conditions are analogous to those given for arithmetic expressions, except that the following hierarchy applies:

arithmetic-expressions
all relational operators
NOT
AND
OR

Parentheses may be used either to improve readability or to override the effects of the hierarchy given above. Each set of conditions within a pair of parentheses is reduced to a single condition. When this is accomplished, reductions which cross parentheses are done.

Examples:

a. Using parentheses for ease of reading.

The following expression

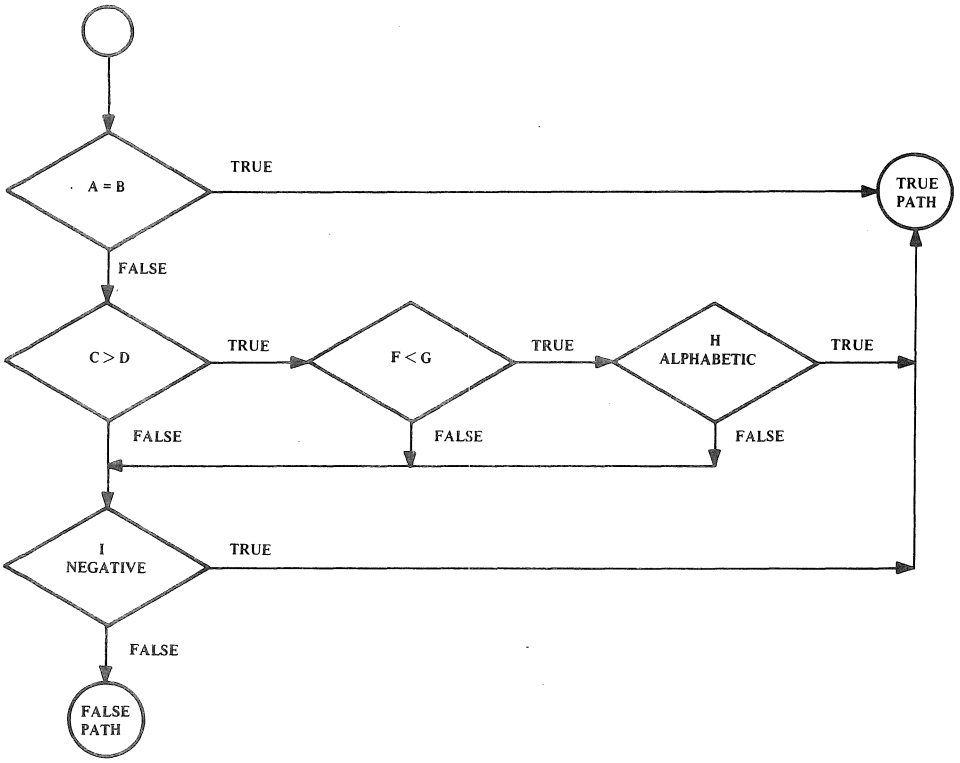
A = B OR C > D AND F < G AND H IS ALPHABETIC OR I IS NEGATIVE

can be parenthesized for readability without changing its effect as shown below.

(A = B) OR (C > D AND F < G AND H IS ALPHABETIC) OR (I IS NEGATIVE)

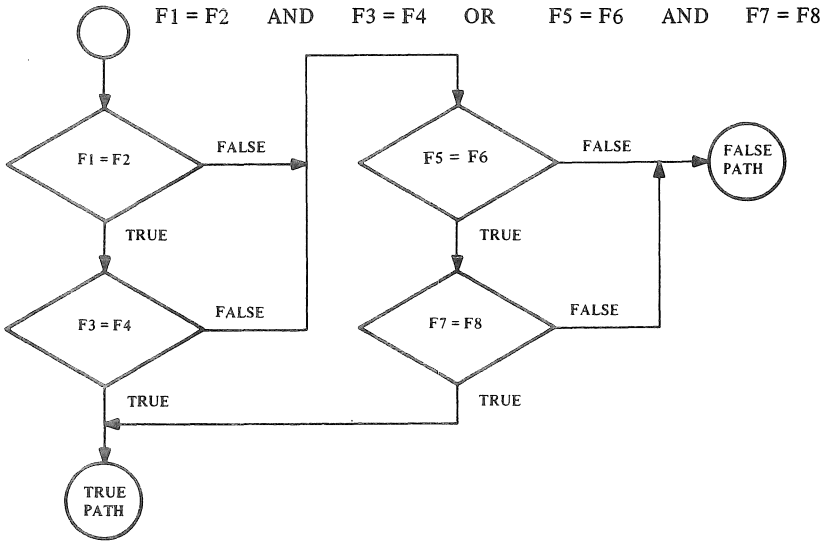
If all the conditions within any of the three sets of parentheses are true, then the entire conditional expression is true.

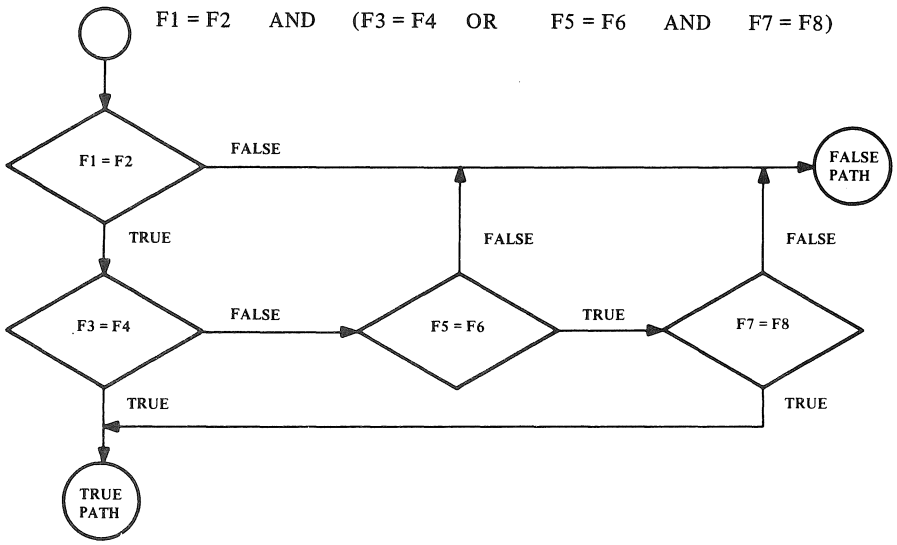
The diagram below illustrates the effect of this statement and the order of evaluation.

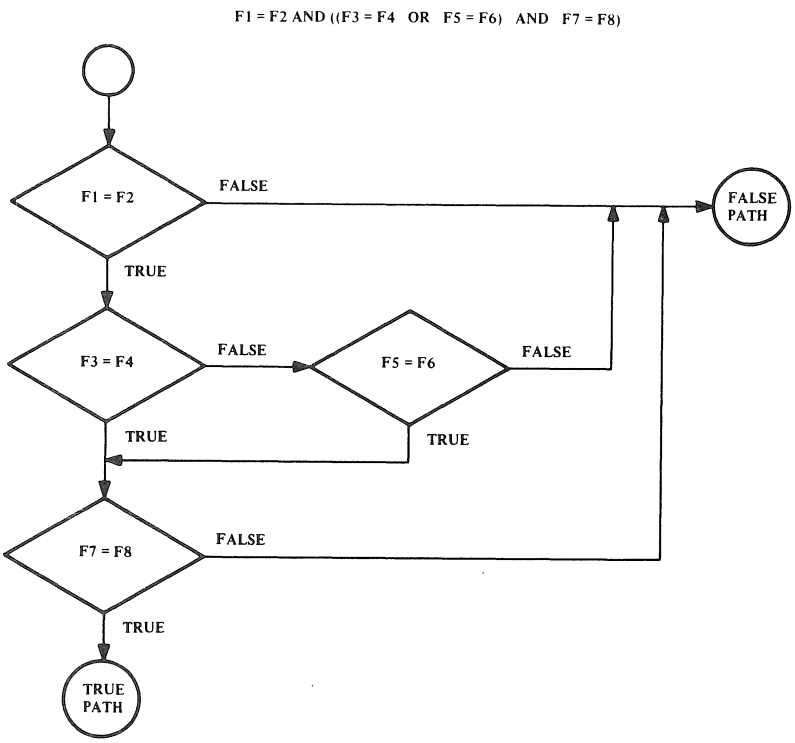


b. Using parentheses to override normal order of evaluation.

To illustrate this usage, a compound-conditional is shown in three forms, each accompanied by a flow diagram showing the result of each.




F1 = F2 AND (F3 = F4 OR F5 = F6 AND F7 = F8)
F1 = F2
FALSE
FALSE PATH
TRUE
FALSE
FALSE
F3 = F4
FALSE
F5 = F6
TRUE
F7 = F8
TRUE
TRUE
TRUE PATH


F1 = F2 AND ((F3 = F4 OR F5 = F6) AND F7 = F8)
F1 = F2
FALSE
FALSE PATH
TRUE
F3 = F4
FALSE
F5 = F6
FALSE
TRUE
TRUE
F7 = F8
FALSE
TRUE
TRUE PATH

6.5.8 Abbreviations in Relation Conditions

When a string of consecutive relation conditions appears in a statement, abbreviations can be used, in certain cases, for any relation condition other than the first. The subject, or the subject and relational operator, or the subject, relational operator and logical connective may be omitted. In each of these cases, the effect of the abbreviated relation condition is as if the omitted parts were the same as those in the nearest preceding complete relation condition within the same sentence. There are three valid forms of abbreviation.

a. Abbreviation 1

If the subject is identical in a series of relational conditions, it can be omitted in all the relational conditions except the first.

Example: A = B OR A < C AND A = D OR A = E

can be abbreviated to

A = B OR < C AND = D OR = E

b. Abbreviation 2

If subjects and relational operators are identical in a series of relational conditions, they can be omitted in all the relational conditions except the first.

Example: A = B OR A = C AND A = D OR A = E

can be abbreviated to

A = B OR C AND D OR E

c. Abbreviation 3

If the subjects, relational operators, and logical connectives are all identical in a series of relational conditions, the subject and relational operator can be omitted in all the relational conditions except the first, and the logical connective can be omitted in all the relational conditions except the last.

Example: A = B AND A = C AND A = D AND A = E

can be abbreviated to

A = B, C, D, AND E

6.6 COMMON OPTIONS ASSOCIATED WITH THE ARITHMETIC VERBS

Associated with the five arithmetic verbs (ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT) are two options: the ROUNDED option, and the ON SIZE ERROR option. These two options are described here to avoid the necessity of including their descriptions with each of the arithmetic verbs.

If the ROUNDED option is specified, the absolute value of the item is increased by 1 if the leftmost truncated digit is greater than 4.

Example:	result:	$567_{\wedge}8756$
	resultant-identifier picture:	999V99
	stored result without ROUNDED option:	$567_{\wedge}87$
	stored result with ROUNDED option:	$567_{\wedge}88$

When the low-order positions in a resultant-identifier are represented by the symbol P in the PICTURE associated with the resultant-identifier, rounding or truncation occurs relative to the rightmost integer position for which storage is allocated.

Example:	result:	5388
	resultant-identifier picture:	99PP
	stored result without ROUNDED option:	53
	stored result with ROUNDED option:	54

6.6.1 The SIZE ERROR Option

If, after decimal point alignment, the number of significant digits in the result of an arithmetic operation is greater than the number of integer positions provided in the result-identifier, a size error condition occurs. Division by zero always causes a size error condition. The size error condition applies to both the intermediate results and the final result of an arithmetic operation. If the ROUNDED option is specified, rounding takes place before checking for size error. When such a size error does occur, the subsequent action depends upon whether or not the SIZE ERROR option is specified.

If the SIZE ERROR is not specified and a size error condition occurs, the value of the resultant-identifier is unpredictable, and no additional action is taken.

If SIZE ERROR is specified, and a size error condition occurs, then the values of the resultant-identifier(s) affected by the size errors are not altered. Values for resultant-identifier(s) for which no size error condition occurs are unaffected by size errors that occur for other resultant-identifier(s). After completion of the execution of the arithmetic operation, the statement after SIZE ERROR is executed.

Example: ADD A TO B ON SIZE ERROR GO TO OVERFLOW.

A:	954
B:	PICTURE IS 999; VALUE 954.
Result:	The contents of B are left unchanged and control is transferred to the paragraph or section named OVERFLOW.

6.7 THE CORRESPONDING OPTION

The CORRESPONDING option is used in the formats of two of the arithmetic verbs (ADD and SUBTRACT) and in the format of the MOVE verb.

For the purpose of this discussion, d_1 and d_2 represent identifiers that refer to group items. A pair of data items, one from d_1 and one from d_2, correspond if the following conditions exist:

a. A data item in d_1 and a data item in d_2 have the same data-name and the same qualification up to, but not including, d_1 and d_2.

b. Both of the data items are elementary numeric data items in the case of an ADD or SUBTRACT statement with the CORRESPONDING option.

Neither d_1 nor d_2 may be data items with level-number 66, 77, or 88.

Each data item subordinate to d_1 or d_2 that contains a RENAMES, a REDEFINES or an OCCURS clause is ignored. However, d_1 and d_2 may have REDEFINES or OCCURS clauses or be subordinate to data items with REDEFINES or OCCURS clauses.

See "ADD," "MOVE," and "SUBTRACT" for information on the specific formats and results of the use of the CORRESPONDING option.

6.8 PROCEDURE DIVISION VERB FORMATS

The format of each PROCEDURE DIVISION verb is given on the following pages. The verbs are presented in alphabetical order.

The word "identifier" is a data-name followed, as required, by any qualification, subscripts, and/or indexes to make the data-name unique.

ACCEPT

Function

The ACCEPT statement causes low-volume data to be read from the user's Teletype console.

General Format

ACCEPT identifier-1 [, identifier-2] ... [FROM mnemonic-name]

Technical Notes

a. The ACCEPT statement causes the next set of data available from the hardware device to replace the contents of the item named by identifier-1, identifier-2,....

b. If the FROM option is specified, the mnemonic-name must appear in the CONSOLE IS clause of the SPECIAL-NAMES paragraph.

ADD

Function

The ADD statement computes the sum of two or more numeric operands and stores the result.

General Format

Option 1

```
ADD { identifier-1 } [ , { identifier-2 } ] ...   TO identifier-m [ROUNDED]
    { literal-1    }     { literal-2    }
    [ , identifier-n [ROUNDED] ] ...
    [ON SIZE ERROR statement.]
```

Option 2

```
ADD { identifier-1 } , { identifier-2 } [ , { identifier-3 } ] ...
    { literal-1    }   { literal-2    }     { literal-3    }

    GIVING identifier-m [ROUNDED]  [ , identifier-n [ROUNDED] ] ...

    [ON SIZE ERROR statement.]
```

Option 3

```
ADD { CORRESPONDING } identifier-1 TO identifier-2
    { CORR          }

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

Technical Notes

a. Each ADD statement must contain at least two operands (i.e., an addend and an augend).

In options 1 and 2, each identifier must refer to an elementary numeric item, except that identifiers appearing to the right of the word GIVING may refer to numeric edited items. In option 3, each identifier must refer to a group item.

Each literal must be a numeric literal; the figurative constant ZERO is permitted.

b. The composite of all operands (i.e., the data item resulting from the superimposition of all operands aligned by decimal point) must not contain more than 19 decimal digits.

c. Option 1 causes the values of the operands preceding the word TO to be algebraically summed. The resultant sum is then added to the current value of identifier-m and this result replaces the current value in identifier-m. If other identifiers follow, the same process is repeated for each of them.

d. Option 2 causes the values of the operands preceding the word GIVING to be algebraically summed. The resultant sum then replaces the current contents of identifier-m. If other identifiers follow, their contents are also replaced by this resultant sum. The current values of identifier-m, identifier-n, ... do not enter into the arithmetic computation.

e. Option 3 causes the data items in the group item associated with identifier-1 to be added to the current value of the corresponding data items associated with identifier-2, and each result replaces the value of the corresponding data-items associated with identifier-2. The criteria used to determine whether two items are corresponding are described under "The CORRESPONDING Option" at the beginning of this chapter.

f. The ROUNDED and ON SIZE ERROR options are described earlier in this chapter under "Common Options Associated with Arithmetic Verbs".

ALTER

Function

The ALTER statement changes the object of one or more GO TO statements.

General Format

```
ALTER   procedure-name-1 TO PROCEED TO procedure-name 2

        [, procedure-name-3 TO PROCEED TO procedure-name-4] ...
```

Technical Notes

a. During execution of the object program, the ALTER statement modifies the GO TO statement in the paragraph named procedure-name-1, procedure-name-3, ... replacing the object of the GO TO by procedure-name-2, procedure-name-4, ..., respectively.

b. Each procedure-name-1, procedure-name-3, ... must be the name of a paragraph that contains only a single GO TO statement without the DEPENDING option.

c. Each procedure-name-2, procedure-name-4, ... must be the name of a paragraph or section within the PROCEDURE DIVISION.

d. A GO TO statement in a section whose priority is greater than or equal to 50 must not be referred to by an ALTER statement in a section with a different priority.

e. An ALTER statement in a procedure not in the DECLARATIVES portion of the program may not reference a procedure name within the DECLARATIVES; conversely, an ALTER statement with the DECLARATIVES may not reference a procedure-name not in the DECLARATIVES.

f. Restrictions similar to those in Note e also apply to the INPUT PROCEDURES and to the OUTPUT PROCEDURES associated with sort verbs.

CLOSE

Function

The CLOSE statement terminates the processing of input and output files, reels, or units.

General Format

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

[ , file-name-1 [ { REEL  } ]  [ WITH { NO REWIND } ] ] ...
                { UNIT  }              { LOCK      }
```

Technical Notes

a. Each file-name must appear as the subject of an FD entry in the FILE SECTION of the DATA DIVISION.

b. The REEL, UNIT, and NO REWIND options apply only to magnetic tape files. UNIT is synonymous with REEL.

c. For the purpose of showing the effect of various CLOSE options as applied to the various storage media, all input, output, and input-output files are divided into the following three mutually exclusive categories:

(1)	NON-REEL	A file whose device is such that the concepts of REWIND, REEL, or UNIT have no meaning. This category includes files residing on disk, punched cards, paper tape, line printer, and Teletype.
(2)	SINGLE-REEL	A file that is entirely contained on one reel or unit.
(3)	MULTI-REEL	A file that may be contained on more than one reel or unit.

The results of each CLOSE option for each of the above types of files are summarized in Table 6-6. The definitions for the symbols used in this table are given below. Where the definition depends upon whether the file is an input or output file, alternate definitions are given; otherwise, the single definition given applies to both input and output files.

A Any subsequent reels of this file will not be processed.

B The current reel is not rewound.

C Standard CLOSE File Procedure

INPUT and I-O Files (see "OPEN")

If the file is positioned at its end, the user's ENDING FILE LABEL PROCEDUREs are performed, if the user has specified any via a USE statement. An input file is considered to be at the end-of-file if the imperative-statement in the AT END clause of a READ for the file has been executed, and no CLOSE statement for the file has been executed.

OUTPUT Files

If LABEL RECORDS are STANDARD, an ending label is created and written on the output medium. Then, any user ENDING FILE LABEL PROCEDUREs are performed.

D The current reel is rewound and unloaded.

E Any attempt to subsequently OPEN this file will result in an error message being typed and the run terminated.

F Standard CLOSE REEL Procedure

INPUT Files

(1) If the file is assigned to more than one device, the next device specified in the ASSIGN clause becomes the current device. If no other device is specified, the first device mentioned becomes the current device.

(2) The standard beginning reel label procedure and the user's BEGINNING REEL LABEL PROCEDURE (specified in a USE statement) are performed for the new reel.

OUTPUT and I-O Files

(1) The standard ending reel label procedure and any user's ENDING REEL LABEL PROCEDURE are performed.

(2) If the file is assigned to more than one device, the devices are swapped. A halt occurs to allow the user to mount an available reel.

(3) The standard beginning reel label procedure and any user's BEGINNING REEL LABEL PROCEDURE are performed.

G The tape is rewound.

X Illegal. This is an illegal combination of a CLOSE option and a file type.

Table 6-3
CLOSE Options and File Types

CLOSE Option	File Type: NON-REEL	File Type: SINGLE REEL/UNIT	File Type: MULTI-REEL
CLOSE	C	C,G	C,G,A
CLOSE WITH LOCK	C,E	C,G,E	C,G,E,A
CLOSE WITH NO REWIND	X	C,B	C,B,A
CLOSE REEL	X	X	F,G
CLOSE REEL WITH LOCK	X	X	F,D
CLOSE REEL WITH NO REWIND	X	X	F,B

d. If a file is OPENed but not CLOSEd before the STOP RUN statement is executed, the file will be automatically CLOSEd.

e. If the file has been specified with an OPTIONAL clause in the FILE-CONTROL paragraph of the ENVIRONMENT DIVISION and the file was not present for this run, the CLOSE has no effect.

f. If a CLOSE statement without the REEL or UNIT option has been executed for a file, a READ, WRITE, or CLOSE statement for that file must not be executed until another OPEN for that file has been executed.

COMPUTE

Function

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

General Format

```
COMPUTE identifier-1 [ROUNDED] { EQUALS  } { identifier-2          }
                               { EQUAL TO } { literal-1            }
                               { =        } { arithmetic-expression }

    [ ON SIZE ERROR statement. ]
```

Technical Notes

a. The COMPUTE statement allows the user to combine arithmetic operations without the restrictions on the composite of operands and/or receiving data items imposed by the arithmetic statements ADD, SUBTRACT, MULTIPLY, and DIVIDE. If the composite operand exceeds 19 decimal digits, the composite is converted to COMP-1 format.

b. Identifier-1 must be an elementary numeric or numeric edited item.

c. Identifier-2 must be an elementary numeric item. Literal-1 must be a numeric literal.

The identifier-2 and literal-1 options provide a method for setting the value of identifier-1 equal to identifier-2 or literal-1.

d. The rules for forming arithmetic expressions and the order of evaluation are given earlier in this chapter under "Arithmetic Expressions."

e. The ROUNDED and SIZE ERROR options are described earlier in this chapter under "Common Options Associated with the Arithmetic Verbs".

DISPLAY

Function

The DISPLAY statement causes low-volume data to be written on the user's Teletype console.

General Format

```
DISPLAY { literal-1    } [ , { literal-2    } ] ...
        { identifier-1 }     { identifier-2 }

      [ UPON mnemonic-name ]
```

Technical Notes

a. The contents of each operand are written on the user's Teletype console in the order listed.

b. Each of the literals can be numeric or nonnumeric, or one of the figurative constants. If a figurative constant is specified as one of the operands, only a single occurrence of that constant is written on the device.

c. The mnemonic-name must appear in the CONSOLE clause in the SPECIAL-NAMES paragraph of the Environment Division.

DIVIDE

Function

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

General Format

Option 1

```
DIVIDE { identifier-1 } INTO identifier-2 [ROUNDED] [REMAINDER identifier-4]
       { literal-1    }

[ ON SIZE ERROR statement. ]
```

Option 2

```
DIVIDE { identifier-2 } BY identifier-1 [ROUNDED] [REMAINDER identifier-4]
       { literal-2    }

[ ON SIZE ERROR statement. ]
```

Option 3

```
DIVIDE { identifier-1 } INTO { identifier-2 } GIVING identifier-3
       { literal-1    }     { literal-2    }

[ROUNDED] [REMAINDER identifier-4]
[ ON SIZE ERROR statement. ]
```

Option 4

```
DIVIDE { identifier-2 } BY { identifier-1 } GIVING identifier-3
       { literal-2    }   { literal-1    }

[ROUNDED] [REMAINDER identifier-4]
[ ON SIZE ERROR statement. ]
```

Technical Notes

a. The value of identifier-1 or literal-1 is divided into the value of identifier-2 or literal-2.

In option 1, the resulting quotient replaces the value of identifier-2. In option 2, the resulting quotient replaces the value of identifier-1. In options 3 and 4, the resulting quotient replaces the value of identifier-3.

b. Each DIVIDE statement must contain two operands (i.e., a dividend and a divisor). Both of these operands (identifier-1 and identifier-2) must refer to elementary numeric items. Identifier-3 may be an elementary numeric or numeric edited item. Each literal-1 or literal-2 must be a numeric literal.

c. The ROUNDED and SIZE ERROR options are described earlier in this chapter under "Common Options Associated with Arithmetic Verbs".

d. If the REMAINDER clause is used, the resulting remainder replaces the value of identifier-4.

ENTER

Function

The ENTER statement allows the execution of MACRO and FORTRAN IV subroutines in conjunction with the COBOL program.

General Format

```
ENTER { MACRO      } external-name
      { FORTRAN-IV }

      [ USING { identifier-1     } ] , { identifier-2     } ] ...
              { literal-1        }       { literal-2        }
              { procedure-name-1 }       { procedure-name-2 }
```

Technical Notes

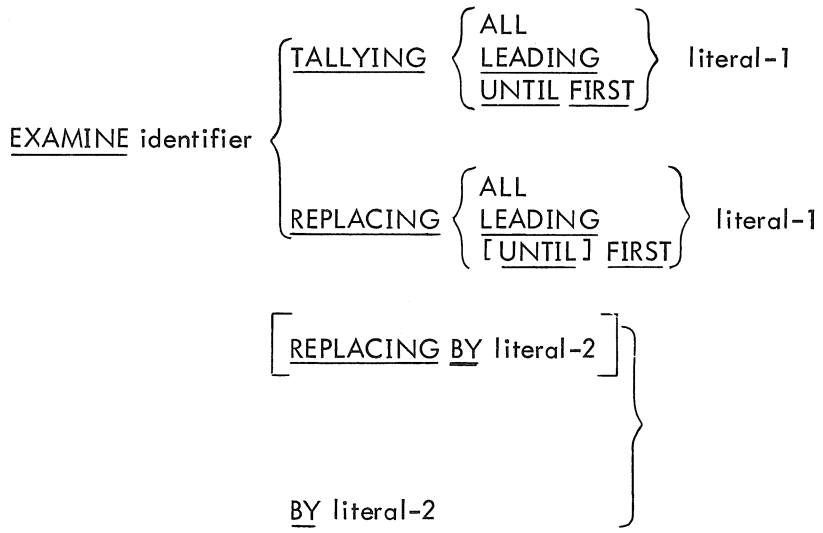
a. The ENTER statement generates a subroutine call followed by the address in which the items associated with the USING clause are located. The ENTER statement is discussed further in Appendix C.

EXAMINE

Function

The EXAMINE statement replaces or counts the number of occurrences of a given character in a data item.

General Format



Technical Notes

a. The USAGE of identifier must be DISPLAY or DISPLAY-7, implicitly or explicitly.

b. Each literal must consist of a single character belonging to a class consistent with that of the identifier. A literal may be any figurative constant.

c. Examination starts at the leftmost character of the identifier and proceeds to the right.

d. When the TALLYING option is used, a count is kept of

(1) Occurrences of literal-1 when the ALL option is used.

(2) Occurrences of literal-1 prior to a character other than literal-1 when the LEADING option is used.

(3) Characters prior to the first occurrence of literal-1 when the UNTIL FIRST option is used.

This count replaces the contents of the special register called TALLY (see "Special Registers," Chapter 1). TALLY has a PICTURE of S99999, and can be referenced in any statement where an identifier referring to an elementary numeric data item is valid.

If the REPLACING BY clause is used with the TALLYING option, replacement is performed according to the rules below.

e. When either of the REPLACING BY options are used, replacement rules are

(1) If the ALL option is used, literal-2 is substituted for each occurrence of literal-1.

(2) If the LEADING option is used, the substitution of literal-2 for literal-1 terminates as soon as a character other than literal-1 is encountered.

(3) If the UNTIL FIRST option is used, literal-2 is substituted for each character prior to the first occurrence of literal-1.

(4) If the FIRST option is used, literal-2 is substituted for only the first occurrence of literal-1.

f. If the identifier is classified as numeric, it must consist solely of numeric characters. It may possess an operational sign, but this sign is ignored by the EXAMINE process.

EXIT

Function

The EXIT statement provides a common end point for a series of routines executed by a PERFORM or USE statement.

General Format

paragraph-name. EXIT.

Technical Notes

a. EXIT must be the first sentence in a paragraph. Only NOTE may follow.

b. The EXIT statement may be used to provide an end point for a series of paragraphs that are PERFORMed, or at the end of a section in the DECLARATIVES. By using EXIT at the end of the range of a PERFORM or USE, a variety of exits from the performed procedure can be accomplished by making each point at which an exit is required a transfer to the EXIT paragraph.

Example:

```
PERFORM TAX-ROUTINE THROUGH EXIT-RTE.
.
.
.
TAX-ROUTINE.
    IF TOTAL-TAX IS EQUAL TO OR GREATER THAN TAX-LIMIT
    GO TO EXIT-RTE.
    MULTIPLY .....
    .
    .
DEDUCTION-RTE.
    IF NO-OF-DEPENDENTS IS EQUAL TO ZERO
    GO TO EXIT-RTE.
    MULTIPLY NO-OF-DEPENDENTS BY DEP-DEDUCT....
    .
    .
EXIT-RTE. EXIT.
```

c. If control reaches an EXIT statement and no associated PERFORM or USE statement is active, control passes through the EXIT paragraph to the first statement of the next paragraph.

GO

Function

The GO TO statement causes control to be transferred from one part of the PROCEDURE DIVISION to another.

General Format

Option 1

```
GO TO [ procedure-name-1 ]
```

Option 2

```
GO TO procedure-name-1, procedure-name-2 [ , procedure-name-3 ] ...
    DEPENDING ON identifier.
```

Technical Notes

a. Each procedure-name is the name of a paragraph or section in the PROCEDURE DIVISION of the program.

b. Option 1 causes transfer of control to the specified procedure-name, or to some other procedure-name if the GO TO has been previously ALTERed.

In order to be alterable, Option 1 must appear as the first sentence in a paragraph; only NOTE may follow.

If procedure-name-1 is not specified, the GO TO must be alterable and an associated ALTER statement must be executed prior to executing this GO TO.

When this form of GO TO appears in an imperative sentence, it must appear as the last or only statement in the sentence, except for NOTE.

c. Option 2 causes transfer of control to procedure-name-1, procedure-name-2, ... or procedure-name-n depending on whether the value of the identifier is 1, 2, ... or n, respectively.

The identifier must refer to an elementary numeric item having no positions to the right of the decimal point. The item may not be USAGE COMPUTATIONAL-1.

If the value of the identifier is other than the positive integers 1, 2, ... or n, the GO TO statement is by-passed.

IF

Function

The IF statement causes a conditional expression to be evaluated and the subsequent operations to be performed to be determined as a result of this evaluation.

General Format

$$\underline{\text{IF}}\ \text{conditional expression}\ \left\{\begin{matrix}\text{statement-1}\\ \underline{\text{NEXT}}\ \underline{\text{SENTENCE}}\end{matrix}\right\}\ \left[;\ \underline{\text{ELSE}}\ \left\{\begin{matrix}\text{statement-2}\\ \underline{\text{NEXT}}\ \underline{\text{SENTENCE}}\end{matrix}\right\}\right]\ .$$

a. Conditional expressions are discussed in Paragraph 6.5 in this chapter.

b. The subsequent action of the program is determined by whether the conditional expression is true or false.

(1) If the conditional expression is true and statement-1 is given, statement-1 is executed and, provided that it does not contain a GO TO or STOP RUN, control passes to the next sentence.

If the conditional expression is true and NEXT SENTENCE is given, control passes to the next sentence.

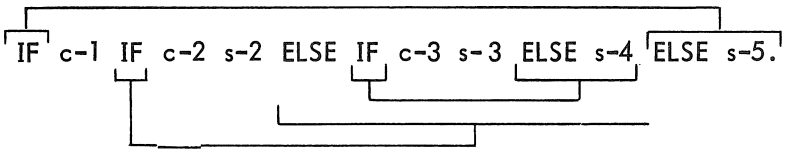
(2) If the conditional expression is false and statement-2 is given, statement-2 is executed and, provided that it does not contain a GO TO or STOP RUN, control passes to the next sentence.

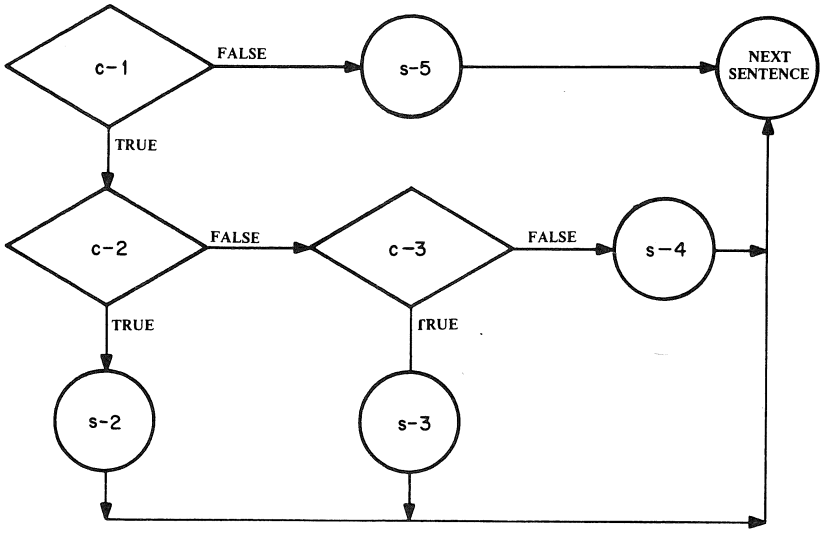
If the conditional expression is false and either ELSE NEXT SENTENCE is given or the entire ELSE clause is omitted, control passes to the next sentence.

c. Statement-1 and statement-2 may be any statement or sequence of statements.

Either statement may contain another IF statement; in this case, this second IF statement is said to be nested. Nested IF statements may be considered paired IF and ELSE combinations. Each subsequent ELSE encountered is considered to apply to the nearest preceding IF that has not already been paired with an ELSE. In other words, the pairing process begins with the innermost nested IF...ELSE pair and works outward.

Example: (c = condition; s = statement)





MOVE

Function

The MOVE statement transfers data in accordance with the rules of editing, from one data area to one or more data areas.

General Format

Option 1

$$\underline{\text{MOVE}} \left\{ \begin{matrix} \text{identifier-1} \\ \text{literal-1} \end{matrix} \right\} \underline{\text{TO}} \text{ identifier-2 } \left[\text{, identifier-3} \right] \ \ldots$$

Option 2

$$\underline{\text{MOVE}} \left\{ \begin{matrix} \underline{\text{CORRESPONDING}} \\ \underline{\text{CORR}} \end{matrix} \right\} \text{ identifier-1 } \underline{\text{TO}} \text{ identifier-2}$$

Technical Notes

a. Identifier-1 (or literal-1) represents the data to be moved and is called the sending item. Identifier-2, identifier-3, ... represent the receiving data items.

b.

c. The following rules apply to both group and elementary items; a group item is treated as a single alphanumeric field.

(1) A numeric edited, alphanumeric edited, or alphabetic data item must not be moved to a numeric or numeric edited data item.

(2) A numeric or numeric edited item must not be moved to an alphabetic data item.

(3) A numeric item whose implicit decimal point is not immediately to the right of the least significant digit must not be moved to an alphanumeric or alphanumeric edited item.

All other moves are legal.

d. The following rules apply to legal moves.

(1) When an alphanumeric, alphanumeric edited, or alphabetic item is the receiving item,

(a) If the size of the sending field is greater than the size of the receiving field, the least significant (rightmost) characters are truncated if the receiving field is not described by a JUSTIFIED RIGHT clause; the most significant (leftmost) characters are truncated if the receiving field is described as JUSTIFIED RIGHT.

(b) If the size of the sending field is less than the size of the receiving field, spaces are placed in the remaining rightmost characters of the receiving field if the receiving field is not described by a JUSTIFIED RIGHT clause; spaces are placed in the remaining leftmost characters of the receiving field if the receiving field is described by a JUSTIFIED RIGHT clause.

(c) If the sizes of the sending and receiving fields are equal, no truncation or filling with spaces takes place.

(2) When a numeric or numeric edited item is the receiving item, the sending and receiving fields are aligned by decimal point. If the sending field is not numeric, the decimal point is assumed to be on the right. Any necessary zero filling takes place before editing. If the receiving item has no operational sign, the absolute value of the sending item is stored. If the receiving item has fewer digits to the left or right of the decimal point than does the sending item, the excess digits are truncated. If the sending item contains any nonnumeric characters, the result is unpredictable.

(3) Any necessary conversion of data from one form of internal representation to another is performed automatically during the move, along with any editing specified by the PICTURE of the receiving item.

e. Any move that is not an elementary move (that is, both the sending and receiving items are not elementary items) is called a group move. A group move is treated as if it were an alphanumeric to alphanumeric elementary move, except that there is no conversion of data from one form of internal representation to another. In other words, the individual data descriptions of the items within the sending group item and the receiving group item are completely ignored and both items are treated as though they were described by a PICUTRE IS X(n) clause, where n is the number of character positions in the particular item.

MULTIPLY

Function

The MULTIPLY statement causes one data item to be multiplied by another data item and the resulting product to be stored in a data item.

General Format

Option 1

```
MULTIPLY { identifier-1 } BY identifier-2 [ROUNDED]
         { literal-1    }

    [ ON SIZE ERROR statement. ]
```

Option 2

```
MULTIPLY { identifier-1 } BY { identifier-2 } GIVING identifier-3
         { literal-1    }    { literal-2    }

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

Technical Notes

a. Each MULTIPLY statement must contain at least two operands (a multiplicand and a multiplier). Each identifier must refer to an elementary numeric item, except that identifier-3 may refer to either a numeric or a numeric edited item. Each literal must be a numeric literal; the figurative constants ZERO and TALLY are permitted.

b. Option 1 causes the value of identifier-1 or literal-1 to be multiplied by the value of identifier-2. The resultant product replaces the value of identifier-2.

c. Option 2 causes the value of identifier-1 or literal-1 to be multiplied by the value of identifier-2 or literal-2. The resultant product replaces the value of identifier-3.

d. The ROUNDED and SIZE ERROR options are described earlier in this chapter under "Common Options Associated with Arithmetic Verbs".

NOTE

Function

The NOTE statement allows the programmer to insert comments in the PROCEDURE DIVISION of his program.

General Format

NOTE character-string .

Technical Notes

a. Any combination of characters from the ASCII character set may be included in the character-string.

b. If the NOTE sentence appears as the first sentence in a paragraph, the entire paragraph is considered to be part of the character-string.

c. If the NOTE statement appears as other than the first sentence in a paragraph, the character-string ends at the first period followed by a space.

d. The contents of the character-string appear on the compilation listing, but are not compiled.

OPEN

Function

The OPEN statement initiates the processing of files and, where necessary, performs the checking and writing of labels.

General Format

```
OPEN { { INPUT  } file-name-1 [WITH NO REWIND] [, file-name-2 [WITH NO REWIND]] ... }...
     { { OUTPUT }                                                                   }
     { { I-O          } file-name-3 [, file-name-4] ...                             }
     { { INPUT-OUTPUT }                                                             }
```

Technical Notes

a. The OPEN statement must be executed for a file prior to the execution of any SEEK, READ, WRITE, or CLOSE for that file.

b. A second OPEN statement for a file cannot be executed prior to the execution of a CLOSE statement for that file.

c. An OPEN statement does not obtain or release the first record of a file. A READ statement must be executed to obtain the first record (or a WRITE statement must be executed to release the first record).

d. More than one file can be opened at a time. The key word INPUT, OUTPUT, INPUT-OUTPUT, or I-O applies to each subsequent file-name until another such key word is encountered or until the end of the OPEN statement is reached.

e. The NO REWIND option has meaning only for magtape files and is ignored for all other devices. If the NO REWIND clause is not specified for a tape file, the tape is rewound to the beginning of tape.

f. If a file has been described as LABEL RECORDS ARE STANDARD, standard label checking or label writing is performed; the user's BEGINNING LABEL (USE) routines are executed if provided. If a file has been described as LABEL RECORDS ARE data-name-1, the user's BEGINNING LABEL (USE) routines are executed. If a file has been described as LABEL RECORDS ARE OMITTED, no label checking or writing is performed.

g. If an INPUT file is described as OPTIONAL (in the FILE-CONTROL paragraph), the operating system will type the message

IS file-name PRESENT?

and wait for the user to type "YES" or "NO". If the user types "NO", the first READ statement for this file causes the imperative-statement at the AT END or INVALID KEY clause to be executed.

h. The I-O or INPUT-OUTPUT options permit the opening of a file on a random-access device for both input and output processing. Since this option requires the existence of a file, it cannot be used when initially creating the file. When the I-O option is specified, the execution of the OPEN statement causes the standard beginning label procedures (see Chapter 8) and the user's BEGINNING LABEL routines, if specified by a USE statement, to be executed.

PERFORM

Function

The PERFORM statement is used to depart from the normal sequence of execution in order to execute one or more procedures and then return control to the normal sequence.

General Format

Option 1

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

Option 2

```
PERFORM procedure-name-1 [THRU procedure-name-2]
        {identifier-1} TIMES
        {integer-1   }
```

Option 3

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

Option 4

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

        VARYING identifier-1 FROM {literal-1   }
                                  {identifier-2}

        BY {literal-2   } UNTIL condition-1
           {identifier-3}

      [ AFTER VARYING identifier-4 FROM {literal-3   }
                                        {identifier-5}
```

BY {literal-4 / identifier-6} UNTIL condition-2

[AFTER VARYING identifier-7 FROM {literal-5 / identifier-8}

BY {literal-6 / identifier-9} UNTIL condition-3]]

Technical Notes

a. Each procedure-name is the name of a section or paragraph in the PROCEDURE DIVISION. Each identifier must refer to a numeric elementary item described in the DATA DIVISION. Each literal must be a numeric literal or the figurative constants ZERO and TALLY.

b. When the PERFORM statement is executed, control is transferred to the first statement of procedure-name-1. An automatic return to the statement following the PERFORM statement is established as follows. The procedures executed constitute the range of the PERFORM.

(1) If procedure-name-1 is a paragraph-name and procedure-name-2 is not specified, the return is after the last statement of procedure-name-1.

(2) If procedure-name-1 is a section-name and procedure-name-2 is not specified, the return is after the last statement in the last paragraph in procedure-name-1.

(3) If procedure-name-2 is a paragraph-name, the return is after the last statement in that paragraph.

(4) If procedure-name-2 is a section-name, the return is after the last statement in the last paragraph of that section.

c. There is no relationship between procedure-name-1 and procedure-name-2, except that the sequence of operations beginning at procedure-name-1 must eventually end with the execution of procedure-name-2 in order to effect the return at the end of procedure-name-2. Any number of GO TO and/or PERFORM statements may occur between procedure-name-1 and procedure-name-2.

d. If control passes to these procedures by means other than a PERFORM statement, control passes through the return point to the following statement as though no return mechanism were present.

e. No perform statement may terminate until all PERFORM statements that it has executed have terminated. No PERFORM statement may be executed which terminates at the same procedure-name as another active PERFORM.

f. Option 1 causes the PERFORM range to be executed once, followed by a return to the statement immediately following the PERFORM.

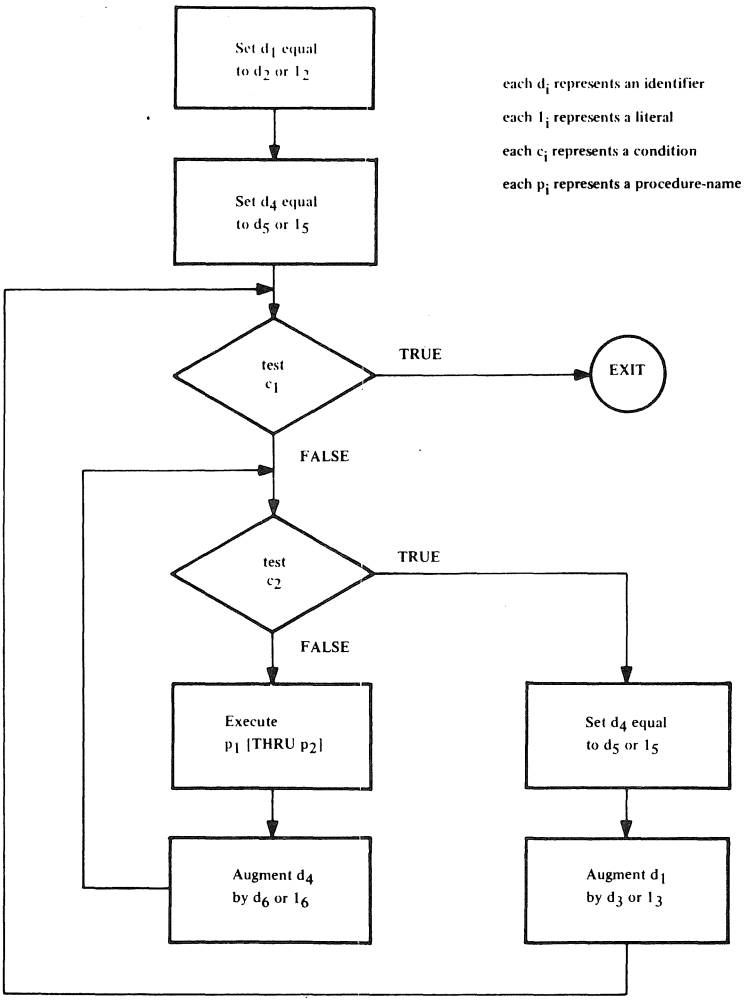
g. Option 2 causes the PERFORM range to be executed the number of times specified by identifier-1 or integer-1. The value of identifier-1 or integer-1 must not be negative; it may be zero. Once the PERFORM statement has been initialized, any modification to the contents of identifier-1 has no effect on the number of times the range is executed.

h. Option 3 causes the PERFORM range to be executed until the condition specified in the UNTIL clause is true. If this condition is true at the time the PERFORM statement is initialized, the range is not executed. Conditions are explained under "Conditional Expressions" earlier in this chapter.

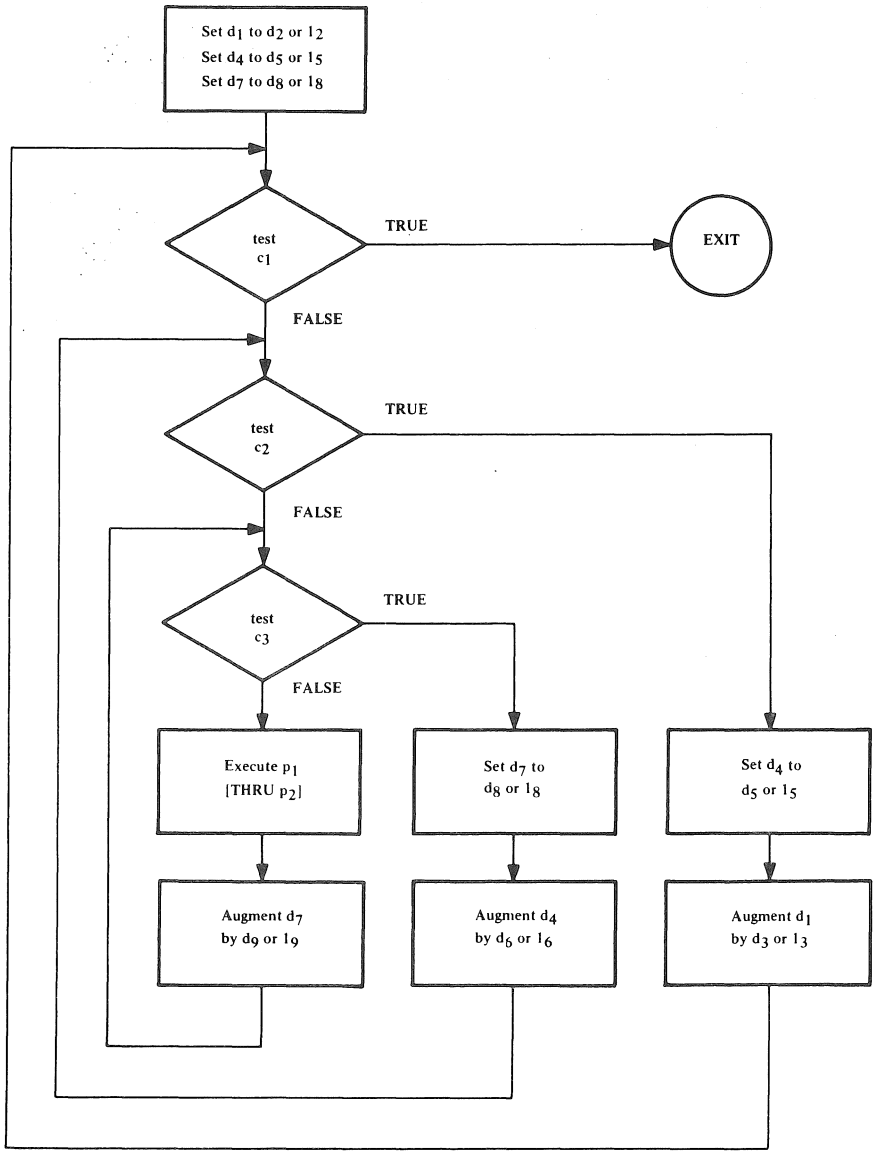
i. Option 4 is used to augment the value of one or more identifiers during the execution of a PERFORM statement.

In option 4, when only one identifier is varied, identifier-1 is set equal to identifier-2 or literal-2 when the PERFORM statement is initialized. If the condition specified is determined to be false at this point, the PERFORM range is executed once. Then the value of identifier-1 is augmented by identifier-3 or literal-3 and the test of the condition is done again. This cycle continues until condition-1 is true; at this point, control passes to the statement following the PERFORM statement. If condition-1 is true at the beginning of the execution of the PERFORM, control immediately passes to the statement following the PERFORM.

The flow chart below illustrates the logic of the PERFORM cycle when two identifiers are varied.


Set d_1 equal to d_2 or l_2
Set d_4 equal to d_5 or l_5
test c_1
TRUE
EXIT
FALSE
test c_2
TRUE
FALSE
Execute p_1 [THRU p_2]
Set d_4 equal to d_5 or l_5
Augment d_4 by d_6 or l_6
Augment d_1 by d_3 or l_3
each d_i represents an identifier
each l_i represents a literal
each c_i represents a condition
each p_i represents a procedure-name

The following flow chart illustrates the logic of the PERFORM cycle when three identifiers are varied.



j. A PERFORM statement that appears in a section whose priority is less than the SEGMENT-LIMIT can have in its range only those procedures contained in sections

(1) Each of which has a priority number less than 50, or

(2) Wholly contained in a single segment whose priority number is greater than 49.

A PERFORM statement that appears in a section whose priority number is equal to or greater than SEGMENT LIMIT, can have in its range only those procedures contained in sections

(1) Each of which has the same priority number as that containing the PERFORM statement, or

(2) Each of which has a priority that is less than the SEGMENT-LIMIT.

When a procedure-name in a segment with a priority number greater than 49 is referred to by a PERFORM statement contained in a segment with a different priority number, the segment referred to is made available in its initial state (that is, with all ALTERable GO TOs set to their initial setting) for each execution of the PERFORM statement.

k. A PERFORM statement in a section not in the DECLARATIVES may have as its range, procedures wholly contained within the DECLARATIVES; however, a PERFORM statement in a section within the DECLARATIVES may not have any non-DECLARATIVE procedures within its range.

l. A PERFORM statement within an INPUT or OUTPUT PROCEDURE associated with a SORT verb may not have within its range any procedures outside of that INPUT or OUTPUT procedure.

READ

Function

The READ statement makes available a logical record from an input file and allows performance of a specified imperative statement when end-of-file or invalid key is detected.

General Format

READ file-name RECORD

[INTO identifier] ; { AT END / INVALID KEY } statement.

Technical Notes

a. An OPEN INPUT or OPEN I-O statement must be executed for the file prior to execution of the first READ statement for that file.

b. The AT END clause is valid only for those files whose ACCESS MODE IS SEQUENTIAL (explicitly or implicitly).

The INVALID KEY clause is valid only for those files whose ACCESS MODE IS RANDOM.

If an end-of-file condition is encountered during the execution of a READ statement for a sequential file, the statement specified in the AT END clause is executed, and no logical record is made available.

The logical end-of-file depends upon the type of device to which the file is assigned

After execution of the imperative-statement in the AT END clause, no further READ statements can be executed for that file without first executing a CLOSE statement followed by an OPEN statement for the file.

If, during the execution of a READ statement for a file whose ACCESS MODE IS RANDOM, the ACTUAL KEY is found to contain a value not within the range specified by the FILE-LIMITS clause for that file, the statement specified in the INVALID KEY clause is executed and no logical record is made available.

c. If a file described by an OPTIONAL clause is not present, the imperative-statement in the AT END or INVALID KEY clause is executed on the first READ for that file. Any specified USE procedures are not performed.

d. If logical end-of-reel is recognized during execution of a READ statement, the following operations are carried out.

(1) The reel is rewound and the user's ENDING LABEL PROCEDUREs are executed, if specified in a USE statement.

(2) If the file is assigned to more than one device, the devices are swapped. The previous reel is rewound and the message

MOUNT REEL nn OF file-name ON device-name

is typed on the user's Teletype console. The program then waits for the operator to type

CONTINUE

after he has mounted the next reel.

(3) The standard beginning label procedure (see Chapter 8) and the user's BEGINNING LABEL PROCEDURE are executed, if specified in a USE statement.

(4) The first data record on the new reel is made available.

e. If a file consists of more than one type of logical record, these records automatically share the same storage area. This is equivalent to an implied REDEFINE for the record area. Only information in the current record is accessible.

f. If the INTO identifier option is specified, the READ statement is then equivalent to a READ without the INTO option, followed by a MOVE of the record area associated with the file-name to identifier.

RELEASE

Function

The RELEASE statement transfers records to the initial phase of the sort operation.

General Format

RELEASE record-name [FROM identifier]

Technical Notes

a. A RELEASE statement may be used only in an input procedure associated with a SORT statement for a file whose SD description contains record-name.

b. If the FROM option is used, the contents of identifier are moved to record-name, then the contents of record-name are released to the sort subroutines.

c. After the RELEASE statement is executed, the contents of record-name may no longer be available.

RETURN

Function

The RETURN statement obtains sorted records from the final phase of a sort operation.

General Format

RETURN file-name RECORD [INTO identifier]; AT END statement.

Technical Notes

a. File-name must be described by an SD file descriptor.

b. A RETURN statement may be used only in an output procedure associated with a SORT statement for file-name.

c. If the INTO phrase is specified, the current record is moved from the record area associated with file-name to identifier.

d. After executing the statement in the AT END clause, no RETURN statements may be executed until another SORT is executed.

SEEK

Function

The SEEK statement initiates the accessing of a mass storage data record for subsequent reading or writing.

General Format

SEEK file-name RECORD

Technical Notes

a. The SEEK statement uses the contents of the ACTUAL KEY item to position the read-write arms on a mass storage device. If the key is invalid, no action is taken; however, if the contents of ACTUAL KEY are not changed before the next READ or WRITE statement is executed, that READ or WRITE statement will then take the INVALID KEY path.

b. The file must be assigned to a mass-storage device, and the ACCESS MODE must be RANDOM.

SET

Function

The SET statement allows a data-item to be incremented, decremented, or set to a value.

General Format

$$\underline{\text{SET}}\ \text{identifier-1}\ [\,,\ \text{identifier-2}]\ldots \left\{\begin{array}{l}\underline{\text{TO}}\\ \underline{\text{UP BY}}\\ \underline{\text{DOWN BY}}\end{array}\right\} \left\{\begin{array}{l}\text{identifier-3}\\ \text{literal-1}\end{array}\right\}$$

Technical Notes

a. All identifiers must be numeric elementary items described without any positions to the right of the assumed decimal point.

All literals must be integers, or the figurative constant ZERO.

b. The SET statement causes identifier-1, identifier-2, ... to be set (TO), incremented (UP BY), or decremented (DOWN BY) the value of identifier-3 or literal-1.

SORT

Function

The SORT statement provides the capability of ordering a file of records according to a set of user-specified keys within a record.

General Format

```
SORT file-name-1 ON { ASCENDING  } KEY data-name-1
                    { DESCENDING }

    [ , data-name-2] ... [; ON { ASCENDING  } KEY data-name-3
                               { DESCENDING }

        [ , data-name-4] ...] ...

{ ; INPUT PROCEDURE IS  procedure-name-1 [ THRU procedure-name-2] }
{ ; USING file-name-2                                             }

{ ; OUTPUT PROCEDURE IS  procedure-name-3 [ THRU procedure-name-4] }
{ ; GIVING file-name-3                                             }
```

Technical Notes

a. File-name-1 must be described in an SD file description entry in the Data Division. Each data-name must represent data items described in records associated with file-name-1.

b. File-name-2 and file-name-3 must be described in an FD file description, not an SD file description, in the Data Division. All records associated with file-name-2 must be large enough to contain all of the KEY data-names.

c. The data-names following the word KEY are listed in order of decreasing significance without regard to how they are divided into KEY clauses.

d. The data-names may be qualified but not subscripted.

e. SORT statements may appear anywhere in the Procedure Division except in the Declaratives portion or in an input or output procedure associated with a sort.

f. When the ASCENDING clause is used, the sorted sequence is from the lowest value to the highest value; when a DESCENDING clause is used, the sorted sequence is from the highest value to the lowest value.

g. The input procedure, if present, must consist of one or more sections or paragraphs that appear contiguously in a source program and do not form a part of any output procedure. The input procedure must contain at least one RELEASE statement in order to transfer records to the sort subroutine.

h. The output procedure, if present, must consist of one or more sections or paragraphs that appear contiguously in a source program and do not form a part of any input procedure. The output procedure must contain at least one RETURN statement in order to make sorted records available for processing.

i. ALTER, GO and PERFORM statements in the input procedure are not permitted to refer to procedure-names outside the input procedures; similarly, ALTER, GO and PERFORM statements in the output procedure are not permitted to refer to procedure-names outside the output procedures.

j. If an input or output procedure is specified, those procedures are PERFORMed by the SORT statement, and all rules relating to the range of a PERFORM must be observed.

k. If the USING option is specified, all the records in file-name-2 are automatically transferred to the sort subroutine. At the time of the execution of the SORT statement, file-name-2 must not be open. Any USE procedures associated with file-name-2 will be executed as appropriate.

The USING option is equivalent to the following input procedure:

```
%1. OPEN INPUT file-name-2
%2. READ file-name-2 INTO sort-record; AT END GO TO %3.
    RELEASE sort-record.
    GO TO %2.
%3. CLOSE file-name-2.
```

l. If the GIVING option is specified, all the sorted records in file-name-1 are automatically transferred to file-name-3. At the time of the execution of the SORT statement, file-name-3 must not be open. Any USE procedures associated with file-name-3 will be executed as appropriate.

The GIVING option is equivalent to the following output procedure:

```
%4. OPEN OUTPUT file-name-3.
%5. RETURN sort-file INTO record-name-3; AT END GO TO %6.
    WRITE record-name-3.
    GO TO %5.
%6. CLOSE file-name-3.
```

STOP

Function

The STOP statement halts the object program.

General Format

STOP { literal / RUN }

Technical Notes

a. If the literal option is used, the literal is displayed on the user's Teletype console, and the program waits for the operator to type

CONTINUE

Following receipt of this message, the program continues execution at the statement following the STOP.

The literal may be a figurative constant; in this case, a single character is displayed.

b. If the RUN option is used, all files currently open are closed, and execution of the program is terminated.

SUBTRACT

Function

The SUBTRACT statement is used to subtract one, or the sum of two or more, numeric items from one or more numeric items and set the values of one or more items to the result.

General Format

Option 1

```
SUBTRACT { identifier-1 } [ , { identifier-2 } ] ...
         { literal-1    }     { literal-2    }

    FROM identifier-m [ ROUNDED ] [ , identifier-n [ ROUNDED ] ] ...

  [ ON SIZE ERROR statement. ]
```

Option 2

```
SUBTRACT { identifier-1 } [ , { identifier-2 } ] ...
         { literal-1    }     { literal-2    }

    FROM { identifier-m } GIVING identifier-n [ ROUNDED ]
         { literal-m    }

  [ identifier-p [ ROUNDED ] ] ...

  [ ON SIZE ERROR statement. ]
```

Option 3

```
SUBTRACT { CORRESPONDING } identifier-1 FROM identifier-2
         { CORR          }

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

Technical Notes

a. Each SUBTRACT statement must contain at least two operands (i.e., a subtrahend and a minuend).

In options 1 and 2, each identifier must refer to an elementary numeric item, except that identifiers to the right of the word GIVING may refer to numeric edited items. In option 3, each identifier must refer to a group item.

Each literal must be a numeric literal or the figurative constant ZERO.

b. The composite of all operands (i.e., the data item resulting from the superimposition of all operands aligned by decimal point) must not contain more than 19 decimal digits.

c. Option 1 causes the values of the operands preceding the word FROM to be added together, and this sum to be subtracted from the values of identifier-m, identifier-n, etc.

d. Option 2 causes the values of the operands preceding the word FROM to be added together, the sum subtracted from identifier-m or literal-m, and the result stored as the new values of identifier-n, identifier-p, etc. The current values of identifier-n, identifier-p, etc., do not enter into the computation.

e. Option 3 causes the data items in the group item associated with identifier-1 to be subtracted from and stored into corresponding data items in the group item associated with identifier-2. The criteria used to determine whether two items are corresponding are described under "The CORRESPONDING Option" at the beginning of this chapter.

f. The ROUNDED and SIZE ERROR options are described earlier in this chapter under "Common Options Associated with Arithmetic Verbs".

USE

Function

The USE statement specifies procedures for input-output label and error handling that are in addition to the standard procedures provided.

General Format

Option 1

```
USE AFTER STANDARD ERROR PROCEDURE ON { file-name-1 }
                                      { INPUT      }
                                      { OUTPUT     }
                                      { I-O        }
                                      { INPUT-OUTPUT } .
```

Option 2

```
USE { BEFORE } STANDARD [ { BEGINNING } ] [ { REEL } ]
    { AFTER  }           [ { ENDING    } ] [ { FILE } ]
                                             { UNIT } ]

LABEL PROCEDURE ON { file-name-1 }
                   { INPUT      }
                   { OUTPUT     }
                   { I-O        }
                   { INPUT-OUTPUT } .
```

Technical Notes

a. USE statements may appear only in the DECLARATIVES portion of the PROCEDURE DIVISION.

The DECLARATIVES portion follows immediately after the PROCEDURE DIVISION header and begins with the word

DECLARATIVES.

The DECLARATIVES portion ends with the words

END DECLARATIVES.

Following this must be a section-header as the first entry of the main portion of the PROCEDURE DIVISION.

The DECLARATIVES portion itself consists of USE sections, each consisting of a section-header, followed by a USE statement, followed by the associated procedure paragraphs.

The general format for the DECLARATIVES portion is given below.

```
PROCEDURE DIVISION.
DECLARATIVES.
section-name-1 SECTION.  USE ......
paragraph-name-1a.  (statement)
[paragraph-name-1b.  (statement)]
.
.
[section-name-2 SECTION.  USE ......]
.
.
.
END DECLARATIVES.
section-name-m SECTION.
```

b. The USE statement may follow on the same line as the section-header and must be terminated by a period followed by a space. The remainder of the section must consist of one or more procedural paragraphs that define the procedures to be used.

c. The USE statement itself is never executed, rather it defines the conditions calling for the execution of the USE procedures.

d. The designated procedures are executed at the appropriate time as follows (see also Chapter 8):

(1) Format 1 causes the designated procedures to be executed after completing the standard input-output error routine.

(2) Format 2 causes the designated procedures to be executed at one of the following times, depending upon the options used.

(a) Before or after a beginning or ending input label check procedure is executed.

(b) Before a beginning or ending output label is created.

(c) After a beginning or ending output label is created, but before it is written on tape.

(d) Before or after a beginning or ending input-output label check procedure is executed.

e. There must not be any references to any non-DECLARATIVES procedure within a USE procedure. Conversely, there must be no reference to procedure-names that appear within the DECLARATIVES portion in the non-DECLARATIVES portion, except that PERFORM statements may refer to a USE section or to a procedure contained entirely within such a USE section.

f. Format 1 causes the associated procedures to be executed after the standard input-output error routine has been executed. If the INPUT option is used, the procedures will be executed for all INPUT files; if the OUTPUT option is used, they will be executed for all OUTPUT files; if the I-O or the INPUT-OUTPUT option is used, they will be executed for all INPUT-OUTPUT files; if the file-name-1 option is used, they will be executed only for that particular file.

g. Format 2 causes the associated procedures to be executed at the appropriate times, as indicated by the options selected (see note d and Chapter 8). If the words BEGINNING or ENDING are not included in Format 2, the designated procedures are executed for both the beginning and ending labels. If neither UNIT, REEL, nor FILE is included, the designated procedures are executed for both REEL (or UNIT) labels and for FILE labels.

If the INPUT, OUTPUT, INPUT-OUTPUT, or I-O option is specified, the label procedure applies to every file of that type except those files described as LABEL RECORDS ARE OMITTED.

If the file-name-1 option is used, its file description must not contain a LABEL RECORDS ARE OMITTED clause.

h. Within a given format, a file-name must not be referred to, either implicitly (i.e., by an INPUT, OUTPUT, INPUT-OUTPUT, or I-O option) or explicitly (i.e., by a file-name-1 option), in more than one USE statement.

i. The following chart indicates the valid combinations of USE formats and file types.

WRITE

Function

The WRITE statement transfers a logical record to an output file.

General Format

Option 1

```
WRITE record-name-1 [ FROM identifier-1 ]
[ { BEFORE } ADVANCING { identifier-2 LINES }  ]
[ { AFTER  }           { integer-1 LINES    }  ]
                       { mnemonic-name      }
```

Option 2

```
WRITE record-name-1 [ FROM identifier-1 ] ; INVALID KEY statement.
```

Technical Notes

a. An OPEN OUTPUT or OPEN I-O or OPEN INPUT-OUTPUT statement must be executed for the file prior to the execution of the WRITE statement.

b. After the WRITE is executed, the data in record-name-1 may no longer be available.

c. Record-name-1 must be the name of a logical record in a DATA RECORDS clause of the FILE SECTION of the DATA DIVISION.

d. Format 1 is valid for any file currently open for output, with ACCESS MODE IS SEQUENTIAL.

The ADVANCING clause allows control of the vertical positioning of the paper form for print files as follows.

(1) If the ADVANCING clause is not specified and the recording mode is ASCII (see Chapter 8), BEFORE ADVANCING 1 LINE is assumed.

(2) If identifier-2 or integer-1 is specified, it must represent a positive integer or zero. The form is advanced the number of lines equal to the value of identifier-2 or integer-1.

(3) If mnemonic-name is specified, the form is advanced until the specified channel is encountered on the paper-tape format control loop. Mnemonic-name must be defined by a CHANNEL clause in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION.

(4) If the BEFORE option is used, the record is printed before the form positioning.

(5) If the AFTER option is used, the record is printed after form positioning occurs, and no form positioning takes place after the printing.

If end-of-reel is encountered while writing on magtape, the WRITE statement performs the following operations.

(1) The user's ENDING LABEL PROCEDURE is executed, if specified by a USE statement.

(2) A file mark is written, and the tape is rewound.

(3) If the file was assigned to more than one tape unit, the units are swapped.

(4) The operating system types the message.

MOUNT AVAILABLE TAPE ON device-name

and waits for the operator to type CONTINUE.

(5) A label is written on the new tape, if labels are not OMITTED, and any user's BEGINNING LABEL PROCEDURE is executed.

e. Format 2 is valid only for random-access files whose ACCESS MODE IS RANDOM.

The imperative-statement in the INVALID KEY clause is executed when an attempt is made to execute a WRITE to a segment outside the range of the FILE-LIMITS of the file.

f. When executing a WRITE statement for a SEQUENTIAL file opened for I-O (or INPUT-OUTPUT), the logical record is placed on the file as the next logical record, if the previous input-output operation was a WRITE, or it replaces the previous record, if the previous input-output operation was a READ.

g. If the FROM option is used, the statement is equivalent to:

MOVE identifier-1 TO record-name-1

WRITE record-name-1 (without the FROM option)

Chapter 7
The COBOL Library

The COBOL Library contains source language entries that are available for inclusion in a user's source program at compile time.

For example, a library entry might consist of a PROCEDURE DIVISION section containing procedure statements associated with a frequently used rate computation. If a programmer wishes to include this computation procedure in his program, he need only write a COPY statement referencing this library entry at the point where he wants this procedure included; it is unnecessary for him to write out the full procedure himself. The effect of the COPY is the same as if the text contained in the library entry were actually written as part of the source program. In addition to PROCEDURE DIVISION entries, ENVIRONMENT DIVISION paragraphs and DATA DIVISION FDs and record descriptions can be copied from the library into a user's source program.

COPY

Function

The COPY statement causes inclusion of a library entry into a COBOL source program at the point where the COPY statement appears.

General Format

```
COPY library-name

[ REPLACING word-1 BY { word-2       }
                      { identifier-1 }
                      { procedure-name-1 }

    [ , word-3 BY { word-4       } ] ... ] .
                  { identifier-2 }
                  { procedure-name-2 }
```

Technical Notes

a. The COPY statement may appear as follows:

(1) In any of the following paragraphs of the ENVIRONMENT DIVISION:

```
OBJECT-COMPUTER.  COPY library-name....
SPECIAL-NAMES.    COPY library-name....
FILE-CONTROL.     COPY library-name....
I-O-CONTROL.      COPY library-name....
```

(2) In place of any clause in the data or file descriptor in the DATA DIVISION.

(3) In a paragraph in the Procedure Division:

```
paragraph-name. [statement-1] COPY statement. [statement-2].
```

statement-1 may not start with NOTE.

b. In the ENVIRONMENT DIVISION, no other statement or clause may appear in the same sentence as the COPY statement.

c. COPY causes the specified library text to be copied from the COBOL Library, and the result of compilation is the same as if the text were actually a part of the source program. The COPYing process is terminated by the end of the library text.

d. Both the COPY statement itself and the statements of the library text, after any specified replacing has been performed, appear on the output listing produced by the compiler.

e. The text in the library entry must not contain any COPY statements.

f. If the REPLACING option is used, each word specified in the format is replaced by the stipulated word, identifier, or procedure-name that is associated with it in the format. That is, word-2, identifier-1, or procedure-name-1 replaces every occurrence of word-1 in the text copied from the library. The library entry itself is not altered.

Word-1 and word-3 may be a data-name, procedure-name, condition-name, mnemonic-name, or file-name.

Word-2 and word-4 may be any COBOL word, including literals but excluding picture-strings.

Example:

```
COPY LIB001 REPLACING ITEM1 BY AMOUNT OF INCOME-REC.

Library entry:

    TAX-CALC.  MULTIPLY ITEM1 BY TAX-RATE GIVING TAX-DUE.....

Result of COPY:

    TAX-CALC.  MULTIPLY AMOUNT OF INCOME-REC BY TAX-RATE
    GIVING TAX-DUE.....
```

Chapter 8
Standard I-O Processing

This chapter describes how the data is structured, stored, accessed, and moved in I-O operations at run-time. Each of the following topics is described and explained in detail.

a. Access mode
b. Recording mode
c. File tables
d. Channel tables
e. Blocking
f. Label records
g. Multiple-file tape
h. SAME AREA clause
i. SAME RECORD AREA clause
j. File-limits

8.1 ACCESS MODE

Each file has either SEQUENTIAL or RANDOM access mode. For random-access files the mode must be specified by means of the ACCESS MODE clause of the FILE-CONTROL paragraph in the ENVIRONMENT DIVISION (see page 4-12).

8.1.1 SEQUENTIAL Mode

In the SEQUENTIAL mode records are accessed in the order in which they appear on the file. Each READ (after the first) brings into memory from the peripheral device the logical record on that device that immediately follows the logical record previously read from or written on that device.

If the file is open for output, each write appends the record to the end of the file. If the file is open for input-output, the write replaces either the record previously read (if the last operation was READ) or the record following the one previously written (if the last operation was WRITE).

8.1.2 RANDOM Mode

The record to be accessed is specified by the contents of the ACTUAL KEY, without regard to the physical characteristics of the device. The following conditions must exist:

a. The device must be a random-access medium.
b. The records must be blocked (the blocking factor may be 1).
c. If the recording mode is DISPLAY-7 (see Section 8.2), the blocking factor must be 1.

The contents of the ACTUAL KEY determine which record, relative to the beginning of the file, is to be read or written. For example, to read the fifth record of a file, the following statements would appear in the source program.

Example:

```
      .
      .
      .
SELECT FILE ASSIGN TO DSK;
       ACCESS MODE IS RANDOM.
       ACTUAL KEY IS FILE-KEY.
      .
      .
      .
MOVE 5 TO FILE-KEY.
READ FILE; INVALID KEY GO TO YOU-LOSE.
      .
      .
      .
```

NOTE

File-key is a computational item defined by the user. It consists of 10 or fewer digits with no decimal places.

8.2 RECORDING MODE

8.2.1 Default Conditions

At both compilation time and run time, assumptions are made regarding the structure, or recording mode, of data as it appears on the external devices. At compilation time, the recording mode for a file is assumed to be DISPLAY-7 if all the data records for that file are described implicitly or explicitly as DISPLAY-7, or if the WRITE verb is used with the ADVANCING option. In all other cases, the recording mode is assumed to be DISPLAY-6. At run time, if the device is found to be other than magnetic tape DECtape, or a random-access device the recording mode is assumed to be DISPLAY-7. Conversions necessary to conform to the USAGE of the records are made automatically by the I-O routines.

8.2.2 DISPLAY-7

DISPLAY-7 records contain a contiguous set of characters. Each record is terminated by a printer control character (ASCII code 12-15, 20-24), or by a string of such characters. The first record in a block starts in the first character position; each succeeding record in that block starts in the first character position following the previous record.

8.2.3 DISPLAY-6

DISPLAY-6 records contain a contiguous set of words. The right half of the first word, supplied by the I-O routines, contains the number of characters in the record. The last word in the record will, if necessary, have filler added to insure that the record ends at a word boundary, so that records always occupy an integral number of words.

8.3 FILE TABLES

There is a file table for each file named in a SELECT statement in the user's program (see SELECT statement on page 4-11). The address of the last file thus named is contained in the right half of the global FILES..

Figure 8-1 shows the structure of a file table. The various fields in that table are described in the paragraph following the figure.

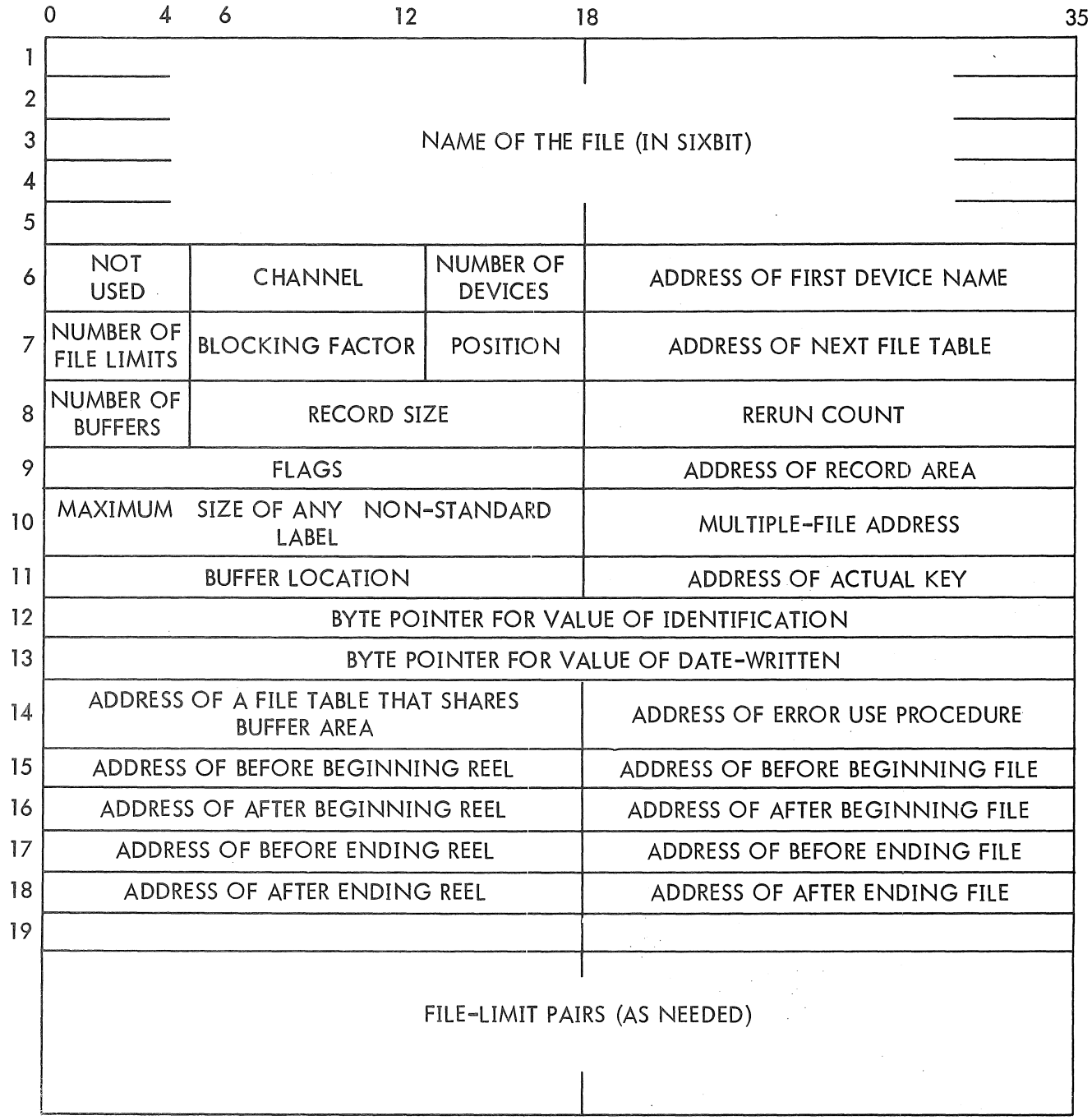


Figure 8.1 Structure of a File Table

8.3.1 Explanation of Fields

NAME OF THE FILE	The name specified by the SELECT clause. It is used only for error messages.
CHANNEL	The relative address of the channel table entry assigned to this file when opened.
NUMBER OF DEVICES	The number of devices specified by the ASSIGN clause (see page 4-11). This number cannot exceed 63.
ADDRESS OF FIRST DEVICE NAME	The address of the first device name in the device-name table. Device names are kept in a table in the order in which they are assigned. After the first name, the addresses of any other assigned device names follow in sequence (i.e., each one is consecutive and contiguous to its predecessor).
NUMBER OF FILE-LIMITS	The number of file-limit pairs associated with the file (see FILE LIMIT clause on page 4-11).
BLOCKING FACTOR	The number of records in a block, as specified by the BLOCK CONTAINS clause (see page 5-7).
POSITION	The position of this file relative to the beginning of a multi-file tape. This position is specified by the POSITION clause in the I-O CONTROL paragraph (see page 4-13).
ADDRESS OF NEXT FILE TABLE	The address of the first word of the next file table. Each file table points to the succeeding table, in reverse order to that in which the files are named in the SELECT statement. The last file table has zeros in this field.
NUMBER OF BUFFERS	The number of buffers requested from the monitor. If the access mode is RANDOM or RELATIVE this number is 1. If the access mode is SEQUENTIAL, this number is 2, unless the user requests a different number of buffers by means of the RESERVE clause (see page 4-11).
RECORD SIZE	The size of the record given in bytes if the recording mode of the file is FIXED, or VARIABLE ASCII. However, if the recording mode is VARIABLE and not ASCII, the size of the largest record is given in words.
RERUN COUNT	The value of the integer, if the RERUN EVERY integer RECORDS clause is given for this file.
FLAGS	A half-word containing flags and fields with the meanings shown in Table 8-1.

Table 8-1
Flags and Fields in File Table

Bits	Meaning
0-1	Type of data on the device: 00 = SIXBIT (DISPLAY-6) 01 = BINARY 10 = ASCII (DISPLAY-7)
2-3	Type of labels: 00 = OMITTED 01 = STANDARD 10 = NON-STANDARD
4	This file is open for input.
5	This file is open for output.
6	This is a random-access device.
7	AT END path has been taken.
8	Device is a console
9	Type of record: 0 = Fixed length 1 = Variable length
10	Rerun is to be taken at end of reel.
11	Rerun is to be taken every now and then (see RERUN COUNT field).
12	File is optional and not present.
13	File is optional.
14-15	Type of data in core: 00 = SIXBIT 01 = BINARY 10 = ASCII
16	Data and Recording modes differ.
17	Access mode: 0 = SEQUENTIAL 1 = RANDOM

ADDRESS OF RECORD AREA

The address of the first location in core memory allocated to a record in this file.

MAXIMUM SIZE OF ANY NON-STANDARD LABEL

The size of the largest label record described in the FD clause.

MULTIPLE FILE ADDRESS

The link between file tables which share a device. If the same device is used by more than one file, the file tables will be linked in a circular, or round-robin, manner through this field.

BUFFER LOCATION

The first location used by the I-O routines for buffers.

ADDRESS OF ACTUAL KEY

The address of the word containing the value to be used as an actual key.

BYTE POINTER FOR VALUE OF IDENTIFICATION

A byte pointer that points to the byte preceding the first character of a string of characters containing the value of identification.

BYTE POINTER FOR VALUE OF DATE-WRITTEN

A byte pointer that points to the byte preceding the first character of a string of characters containing the value of date-written.

ADDRESS OF A FILE TABLE THAT SHARES BUFFER AREA

The link between file tables which share a buffer area. If this file appears in a SAME AREA clause, all associated files in that clause are linked in a circular, or round-robin, manner through this field.

ADDRESS OF ERROR USE PROCEDURE

The address (words 14 through 18) of the first instructions of the USE procedure as declared in the DECLARATIVES (see page 6-58). The compiler generates the following code to call a USE procedure:

```
PERF.  17,%Y
JRST   <use procedure>
POPJ   17,
```

%Y is a location in working storage that will contain the return location for a PERFORM. When the I-O routines want to execute a USE procedure, they pick up the appropriate address in an AC and then execute.

```
PUSHJ  17,(AC)
```

FILE-LIMIT PAIRS

For each file-limit, three words are allocated. The left-half of word 1 contains the address of the lower limit; the right-half of word 1 contains the address of the higher limit. Words 2 and 3 contain the actual values of these limits, at OPEN time.

8.4 CHANNEL TABLES

For each open file, a 10-word entry is placed in a channel table. Table 8-2 shows the contents of each word in that entry.

Table 8-2
Channel Table Entry

Word	Contents
1	Number of bytes per word.
2	Relative number of the current physical block.
3	Number of buffers per logical block.
4	Number of buffers yet to be processed for current block.
5	Number of records required to fill current logical block.
6	IOWD pointing to device-name being used with multi-device file.
7-9	A 3-word header used for output.
10-12	A 3-word header used for input.
13	Number of records remaining until next rerun dump.
14	Current record number.
15	Characteristics of the device, as returned by a DEVCHR UUO.

8.5 BLOCKING

A file is said to be blocked when a BLOCK CONTAINS clause is part of its description; conversely, a file is unblocked when no BLOCK CONTAINS clause is specified. The number of records in a block is termed the blocking factor. The next two paragraphs show how I-O processing differs for blocked and unblocked files.

8.5.1 Reading and Writing Blocked Files

For each READ statement referencing a SEQUENTIAL blocked file, the I-O routines transfer to the record area (i.e., core memory allocated to contain the current record) the next consecutive record from the file. If a number of records, equal to the blocking factor, have been transferred since the previous block was read, the I-O routines do a physical read of the device.

When a RANDOM blocked file is read, both the blocking factor and the ACTUAL KEY are used to specify which physical segment to read and which record in that segment to transfer to the record area.

Writing SEQUENTIAL blocked files is accomplished in a manner similar to the reading of them, except that records are transferred from the record area to the buffer area.

When a RANDOM blocked file is written, the I-O routines do the following:

a. Read a physical segment whose address is determined by the blocking factor and the ACTUAL KEY, if necessary.

b. Transfer the record to a portion of that segment.

c. The segment is written either when another segment is to be read or when the file is closed.

8.5.2 Reading and Writing Unblocked Files

When an unblocked file is either read or written, the I-O routines have complete control over the time when the actual read or write takes place. The user need not concern himself with the physical characteristics of the device; in fact, he does not have to know anything about such characteristics as block or segment sizes.

8.6 LABEL RECORDS

The term label records refers to header or trailer labels on the file. The presence or absence of label records is specified by the LABEL RECORDS clause (see page 5-10). Their format can be standard or non-standard.

8.6.1 Standard Label Records

The standard label for DECtape and random-access devices is the directory block used by the monitor. For magnetic tape, the standard label is 30 characters in length and is written in a separate block from the data, with the same recording made as the data. Table 8-3 shows the contents of each character in a standard label for non-random-access devices.

Table 8-3
Standard Label for Nonrandom-Access Media

Characters	Contents
1-4	HDR1 = Beginning file. EOF1 = Ending file. EOV1 = Ending reel.
5-13	Value of identification.
14-21	Always spaces.
22-27	Not used.
28-31	Reel number. The first reel is always 0001.
32-41	Not used.
42-47	Creation date: two characters each for the year, month, and day, respectively.
48-80	Not used.

8.6.1.1 Ending Labels - Magnetic tapes are the only devices with ending labels. Each ending label is preceded by and followed by an end-of-file mark.

8.6.2 Non-Standard Label Records

Non-Standard labels are specified by the LABEL RECORDS clause (see page 5-10). Similar to standard labels, they are written in a separate block on the device.

When a file is opened, the beginning non-standard label will be read (as input) or written (as output) automatically by the I-O routines. If the file is being opened for output, the data for the record must be supplied by a USE procedure in the DECLARATIVES (see page 6-58). If the file is being opened for input, no checks are made by the I-O routines to determine the validity of the label; the user may write any check in a USE procedure.

8.7 MULTIPLE-FILE TAPE

Only magnetic tapes can contain multiple files in the COBOL sense. This may seem to conflict with the obvious fact that random-access devices contain more than one file, but the programmer will recall that files on a random-access device are treated by the monitor as though they are on separate devices. Each file on a multi-file magnetic tape must have labels, and must be wholly contained on one reel.

8.8 SAME AREA CLAUSE

The SAME AREA clause specifies that two or more files are to share the same memory area during processing. Since the area shared includes all storage areas (including alternate areas) assigned to the files specified in this clause, only one file at a time can be open.

8.9 SAME RECORD AREA CLAUSE

The SAME RECORD AREA clause specifies that two or more files are to use the same memory for processing the current logical record. All or any of the files specified in this clause may be open at any time. The record area contains only one record at any time.

8.10 FILE-LIMITS

File-limits must be specified for RANDOM files. They may also be specified for input SEQUENTIAL files, in which case only that portion of the file that is within the file-limits is read.

File-limits for files whose access mode is RANDOM specify the allowed range, or ranges, for the ACTUAL KEY (see ACTUAL KEY on page 4-12). When the contents of the ACTUAL KEY fall outside all ranges given in the FILE-LIMITS clause for that file, the read or write transfers control to the statement specified in the associated INVALID KEY clause (see pages 6-47 and 6-62).

Appendix A
COBOL Reserved Words

In the listing below, words preceded by no symbols are standard COBOL reserved words that are also reserved in PDP-10 COBOL. Words preceded by a single * are standard reserved COBOL words that are not reserved in PDP-10 COBOL, but should be avoided for compatibility with other COBOL compilers. Words preceded by ** are reserved in PDP-10 COBOL only and should be used for checking programs written for other COBOL compilers.

A

ACCEPT
ACCESS
ACTUAL
ADD
ADVANCING
AFTER
ALL
ALPHABETIC
*ALPHANUMERIC
ALTER
ALTERNATE
AND
*APPLY
ARE
AREA
AREAS
ASCENDING
ASSIGN
AT
AUTHOR

B

BEFORE
BEGINNING
BLANK
BLOCK
BY

C

CH
**CHANNEL
CHARACTERS
CF
*CLOCK-UNITS
CLOSE
COBOL
*CODE
*COLUMN
COMMA
COMP
**COMP-1
COMPUTATIONAL
**COMPUTATIONAL-1
COMPUTE
CONFIGURATION
**CONSOLE
CONTAINS
*CONTROL
*CONTROLS
COPY
CORR
CORRESPONDING
CURRENCY

D

DATA
DATE-COMPILED
DATE-WRITTEN
*DE
DECIMAL-POINT
DECLARATIVES
DEPENDING
DESCENDING

*DETAIL
DISPLAY
**DISPLAY-6
**DISPLAY-7
DIVIDE
DIVISION
DOWN

E

ELSE
END
ENDING
ENTER
ENVIRONMENT
EQUAL
EQUALS
ERROR
EVERY
EXAMINE
EXIT

F

FD
FILE
FILE-CONTROL
FILE-LIMIT
FILE-LIMITS
FILLER
*FINAL
FIRST
*FOOTING
FOR
FROM

G

*GENERATE
GIVING
GO
GREATER
*GROUP

H

*HEADING
HIGH-VALUE
HIGH-VALUES
*HOLD

I

I-O
I-O-CONTROL
IDENTIFICATION
IF
IN
INDEX
INDEXED
*INDICATE
*INITIATE
INPUT
INPUT-OUTPUT
INSTALLATION
INTO
INVALID
IS

J

JUST
JUSTIFIED

K

KEY
KEYS

L

LABEL
*LAST
LEADING
LEFT
LESS
*LIMIT
*LIMITS
LINE
*LINE-COUNTER
LINES
LOCK
LOW-VALUE
LOW-VALUES

M

MEMORY
MODE
MODULES
MOVE
MULTIPLE
MULTIPLY

N

NEGATIVE
NEXT
NO
NOT
NOTE
*NUMBER
NUMERIC

O

OBJECT-COMPUTER
OCCURS
OF
OFF
OMITTED
ON
OPEN
OPTIONAL
OR
OUTPUT

P

*PAGE
*PAGE-COUNTER
**PDP-6
**PDP-10
PERFORM
*PF
*PH
PIC
PICTURE
*PLUS
POSITION
POSITIVE
PROCEDURE
PROCEED
*PROCESS
PROCESSING
PROGRAM-ID

Q

QUOTE
QUOTES

R

RANDOM
*RD
READ
RECORD
RECORDS
REDEFINES
REEL
RELEASE
REMAINDER
REMARKS
RENAMES
REPLACING
*REPORT
*REPORTING
*REPORTS
RERUN
RESERVE
*RESET
RETURN
*REVERSED
REWIND
*RF
*RH
RIGHT
ROUNDED
RUN

S

*SA
SAME
SD
*SEARCH
SECTION
SECURITY
SEEK
SEGMENT-LIMIT
SELECT
SENTENCE
SEQUENTIAL
SET
*SIGN
SIZE
SORT
*SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
STATUS
STOP
SUBTRACT
*SUM
**SWITCH
SYNC
SYNCHRONIZED

T

TALLY
TALLYING
TAPE
*TERMINATE
THAN
THROUGH
THRU
TIMES
TO
**TODAY
*TYPE

U

UNIT
UNTIL
UP
UPON
USAGE
USE
USING

V

VALUE
VALUES
VARYING

W

WHEN
WITH

WORDS

WORKING-STORAGE

WRITE

Z

ZERO

ZEROES

ZEROS

Appendix B
Character Collating Sequence

The following table gives the collating sequence for ASCII (DISPLAY-7) fields as used in condition comparisons.

ASCII	Character	ASCII	Character	ASCII	Character	ASCII	Character
040	blank	070	8	120	P	150	h
041	!	071	9	121	Q	151	i
042	"	072	:	122	R	152	j
043	#	073	;	123	S	153	k
044	$	074	<	124	T	154	l
045	%	075	=	125	U	155	m
046	&	076	>	126	V	156	n
047	'	077	?	127	W	157	o
050	(	100	@	130	X	160	p
051	)	101	A	131	Y	161	q
052	*	102	B	132	Z	162	r
053	+	103	C	133	[	163	s
054	,	104	D	134	\	164	t
055	-	105	E	135	]	165	u
056	.	106	F	136	↑	166	v
057	/	107	G	137	←	167	w
060	0	110	H	140	`	170	x
061	1	111	I	141	a	171	y
062	2	112	J	142	b	172	z
063	3	113	K	143	c	173	{
064	4	114	L	144	d	174	|
065	5	115	M	145	e	175	}
066	6	116	N	146	f	176	ALT MODE
067	7	117	O	147	g	177	DEL

Appendix C
The ENTER PROCEDURE

The ENTER verb is used for linkage to subroutines external to the COBOL program. These subroutines may be written in either MACRO or FORTRAN-IV language.

If the ENTER verb is ENTER MACRO, the subroutine linkage is

```
PUSHJ 17, user-routine
```

If the ENTER verb is ENTER FORTRAN-IV, the subroutine linkage is

```
JSA 16, user-routine
```

If the USING clause appears, a single parameter for each identifier or literal follows the subroutine linkage. (ARG is a PDP-10 instruction that does nothing.)

a. For 1-word COMP, index data items, and index-names:

```
ARG 0, identifier
```

b. For 2-word COMP:

```
ARG 11, identifier
```

c. For COMP-1:

```
ARG 2, identifier
```

d. For DISPLAY-6 or DISPLAY-7:

```
ARG 10, byte-pointer
```

("byte-pointer" contains a byte pointer to the identifier)

e. For Procedure-names:

```
ARG 17, procedure-name
```

C.1 EXAMPLES OF CALL PROCEDURES

Example 1

```
.
.
.
77 FIELD1 PICTURE S9(6) COMP.
77 FIELD2 USAGE INDEX.
77 FIELD3 PICTURE S9(15) COMP.
77 FIELD4 PICTURE XX; DISPLAY-7.
77 FIELD5 PICTURE XX.

01 DUMMY.
    02 FIELD6 OCCURS 3 TIMES INDEXED BY FIELD7.
    .
    .
    .
ENTER MACRO ROUTIN USING FIELD1, FIELD2, FIELD3, FIELD4, FIELD5, FIELD7.
```

The preceding coding will generate:

```
        PUSHJ    17,ROUTIN
        ARG      0,FIELD1
        ARG      0,FIELD2
        ARG      11,FIELD3
        ARG      10,%LIT
        ARG      10,%LIT+1
        ARG      0,FIELD7
    .
    .
    .
%LIT:   POINT    7,FIELD4
        POINT    6,FIELD5
```

Example 2

```
PROCEDURE DIVISION.
ONLY SECTION.
PARA-NAME.  ENTER FORTRAN-IV ROUTIN.
      .
      .
      .
```

The preceding will generate:

```
JSA          16, ROUTIN
```

APPENDIX D

THE COBOL COMMAND

D.1 COMMAND FORMAT

$$\text{COBOL}\left(\begin{matrix}\text{BIN}\\ \text{NOBIN}\end{matrix}\right), \left.\begin{matrix}\text{LIST}\\ \text{NOLIST}\end{matrix}\right., \text{MACRO}, \text{MAP}, \left.\begin{matrix}\text{STD}\\ \text{NONSTD}\end{matrix}\right.)$$

{IN=}filename-1, {BIN=}filename-2, {LST=}filename-3

COBOL places the COBOL compiler into execution, to compile programs written in the COBOL programming language.

BIN produces a binary file suitable for execution.

NOBIN suppresses the production of the binary file. This facility is useful when only a compilation listing of the program is required.

LIST will give a listing of the source program as the file is compiled.

NOLIST will suppress the program listing although the file will be compiled.

MACRO produces a listing of the code generated by the program. This will be listed after the source listing.

MAP produces a listing showing the parameters of each user-defined item. This is listed after the source listing.

STD indicates that the file has sequence numbers in the source program.

NONSTD indicates that the source program does not have sequence numbers. This is generally the case with program files prepared via the Teletype.

'filename-1' is the name of the source file.

'filename-2' is the name of the binary file.

'filename-3' is the name of the listing file.

D-2 GENERAL DESCRIPTION

The source file is input to the COBOL compiler. This produces an output file unless the option NOBIN was specified.

If the input file is omitted then it is assumed to come from the job input device. The end of file must be signified in the usual manner; ↑Z through terminals, a file separator card through Batch.

If the BIN file is omitted then a relocatable file is created bearing the same name as the source file but with a processor program name of REL.

If the LST file is omitted then a list file is sent to the job output device. If the list file has no processor program name specified then LST will be assumed.

D.3 ABBREVIATIONS

COBOL	may be abbreviated to COB or CBL
BIN	may be abbreviated to B
NOBIN	may be abbreviated to NOB
LIST	may be abbreviated to L
NOLIST	may be abbreviated to NOL
MACRO	may be abbreviated to M
NONSTD	may be abbreviated to NS

The minimum command possible is COB or CBL, in this case the file is expected to come from the job input device.

The default options are BIN, NOLIST and STD.

D.4 EXAMPLE

(i)
```
.COBOL(MACRO) COBREC/CBL LST=COBREC/LST<cr>
COBOL:  COBREC

EXIT
↑C

.
```

APPENDIX E

PROGRAMMING IN COBOL

E.1 EFFICIENT COBOL PROGRAMMING ON THE PDP-10

One basic consideration the programmer must remember is that, basically, COBOL is a language that manipulates bytes, whereas the PDP-10 is most efficient when manipulating words.

If a field described in the COBOL program occupies one or more full words, COBOL will try to generate word-move instructions (MOVE,BLT); if a field occupies only part of a word, byte instructions (LDB,DPB) are employed and consequently the program will run more slowly. In addition, when moving data from one field to another, it is best if both fields have the same usage, and both start at the same relative position within a word.

The programmer can ensure alignment of fields by using the SYNCHRONIZED clause in his data description, or by remembering that there are 6 sixbit (DISPLAY-6) bytes, and 5 ASCII (DISPLAY-7) bytes in each PDP-10 machine word, and setting field sizes accordingly.

A second basic consideration the programmer must remember is that COBOL is a decimal language, whereas the PDP-10 is a binary machine. The COMPUTATIONAL usage is meant to alleviate this conflict. COMPUTATIONAL items are stored in binary.

If the programmer describes a numeric field as having usage DISPLAY-6 or DISPLAY-7, COBOL will generate code to convert the data to binary before doing any arithmetic operation. This will not only result in a larger program, it will also be much slower.

For example, take the following items:

```
77  CA   PIC   S99; COMP.
77  CB   PIC   S99; COMP.
77  DA   PIC   S99; DISPLAY-6.
77  DB   PIC   S99; DISPLAY-7.
```

the statement ADD CA TO CB would result in

```
MOVE 1,CA
ADDM 1,CB
```

whereas the statement ADD DA TO DB would result in

```
GD6. 1,[POINT 6,CA]
GD6. 3,[POINT 6,DA]
ADD  1,3
PD6. 1,[POINT 6,CA]
```

(GD6. and PD6. are UUO's which call subroutines to convert from sixbit to binary and back). Execution time for the second example is 100-500 times slower than that for the first example.

E.2 LINKING COBOL PROGRAMS

COBOL programmers interested in calling FORTRAN or MACRO subroutines are referred to Chapter 6 and Appendix C of the COBOL manual for information on the ENTER verb and the calling sequences generated by the COBOL compiler.

It is not possible to create COBOL subroutines. A COBOL program will always be a main program.

APPENDIX F

INPUT-OUTPUT CONSIDERATIONS

F.1 File Formats

F.1.1 Definition of Terms

Logical Record

The smallest unit of data that can be processed by the operating system. In COBOL, this is also called simply RECORD.

Physical Record

The smallest unit of data that can be processed by the hardware (e.g. 128 words for disk, 80 columns for the card reader).

Buffer

An area of core memory into which the monitor reads or from which the monitor writes, a physical record.

Blocking Factor

That number specified in the 'BLOCK CONTAINS' clause of the File-descriptor for the file; if there is no 'BLOCK CONTAINS' clause, the blocking factor is said to be zero.

Logical Block

Those buffers required to contain a number of contiguous records, that number being the blocking factor. A logical block may extend over many buffers, but always uses an integral number of buffers; any ususued portion of the last buffer is wasted. If the smallest record of a file is much smaller than the largest record, there could be several wasted buffers, since the number of buffers required is always determined by the size of the largest record multiplied by the number of logical records contained in the logical block.

File

An ordered collection of contiguous logical records; the largest unit of data that can be processed by the operating system.

F.1.2 Data Structure

A record may be either sixbit (Display-6) or ASCII (Display-7) and either fixed or variable length.

A sixbit record is a set of contiguous words. The first word has, in the right half, the number of characters in the record. The last word may be padded to ensure that the record boundary coincides with a word boundary. The amount of buffer space required is the number of characters in the record plus six characters for the character count in the first word plus the number of padding characters.

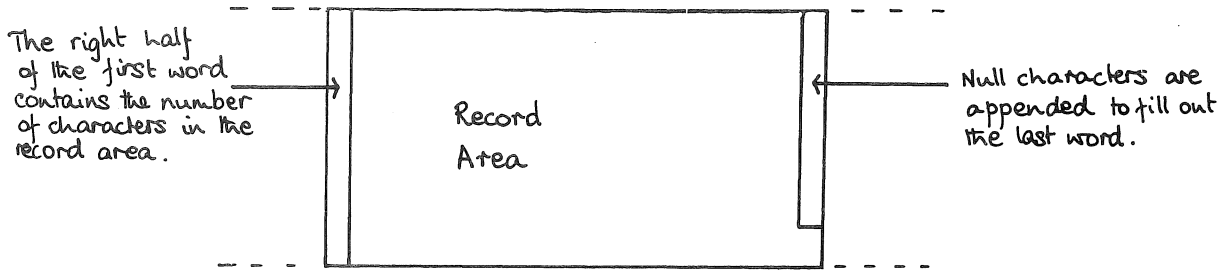


Figure F.1 Sixbit Record

An ASCII record is a set of contiguous characters terminated with a carriage return. If the record was generated with a COBOL WRITE without the advancing clause, a line feed is also appended. If the advancing clause was used a string of up to 63 printer control characters either precedes or follows the record. Word boundaries have no significance, the last character of a record is immediately followed by the first character of the next record. The amount of buffer space required is:

(i) Advancing clause was used; number of characters in the record plus the number of printer control characters plus one for carriage return, or

(ii) Advancing clause was not used; number of characters in the record plus two for carriage return and line feed.

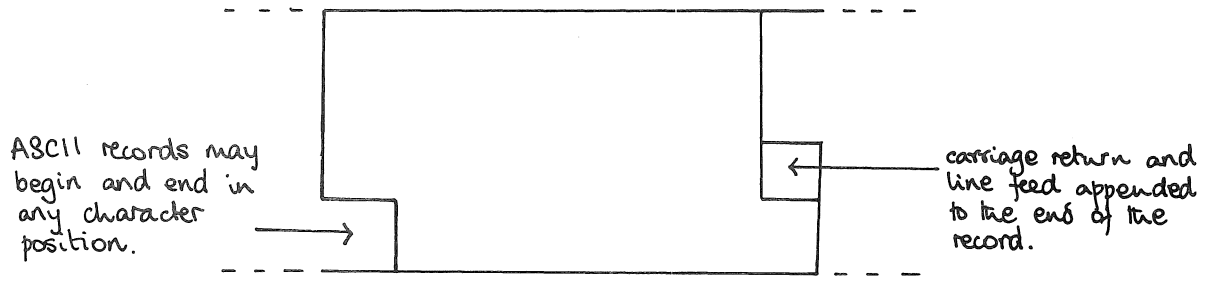


Figure F.2 ASCII Record

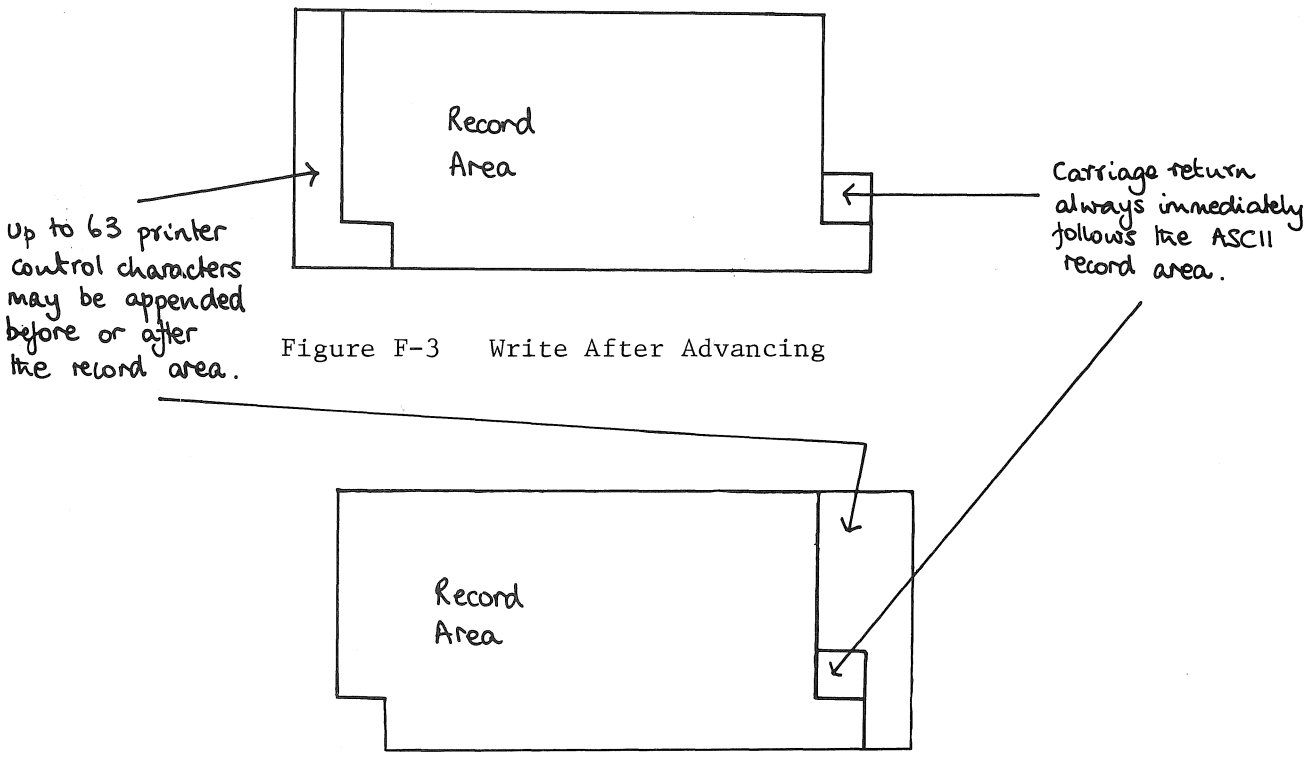


Figure F-3 Write After Advancing

Figure F-4 Write Before Advancing

When reading a record, the operating system recognizes the following as 'end of line' characters;

ASCII codes 12,13,14,15	Line feed, vertical tab, form feed, and carriage return
20,21,22,23,24	Printer control characters
32	Teletype end of file character, ctrl-Z
33,175,176	altmode codes

Leading end of line characters are ignored. A record terminates with the first end of line character or a satisfied character count. If the first character count was satisfied before an end of line character was encountered, the remaining characters are discarded until and end of line character is encountered. If the end of line character comes before the character count is satisfied, ASCII spaces are passed until it is satisfied. ASCII null characters are always ignored.

F.1.3 Blocking

A file is considered *blocked* if the blocking factor is non-zero; it is considered *unblocked* if the blocking factor is zero.

Files are blocked for two reasons:

(i) The output device is a magnetic tape. A non-standard buffer size is used to reduce the number of inter-record-gaps. The non-standard buffer size is set to contain one logical block.

(ii) At some time, the file is to be accessed randomly or the file is to be open for input/output processing. The blocking factor enables the operating system precisely and efficiently to locate a given record.

F.1.4 Labels

Only two devices may have labels written out with the data file, a card file, and a magnetic tape file. Directory devices use the directory for the label. A card file has only a 'beginning-file-label' and it is the first card of the file. A magnetic tape file may have 2 or more labels. If the labeled file is a multi reel file, it has 2 labels for each reel. A labeled file contained entirely on one reel has two labels. See the Table 8-3 for the standard label format.

F.2 Use of Sixbit I/O

DISPLAY-6 (sixbit) files should be used only for applications where speed and efficient use of file storage are important considerations and where file compatibility with Digital software is not a concern. Sixbit input/output is handled by COBOL only. Sixbit input/output will not be handled by any system programs, editors, or symbiont programs.

INDEX

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

COMMENTS

Title of Publication: PDP-10 COBOL REFERENCE MANUAL MNT-13

The Computer Centre welcomes any suggestions that will assist in improving their publications. Please comment on the usefulness and readability of this manual. Suggest additions and deletions and indicate any specific errors and omissions. Please provide page and section references where relevant.

Name : _____

Position : _____

Address : _____

ERRORS

COMMENTS

Please return to the Technical Writer,
Computer Centre,
University of Queensland,
St Lucia,
QUEENSLAND 4067.