IBM Systems Reference Library

IBM DOS Full American

National Standard COBOL

Program Numbers: (Versions 1 & 2) 360N-CB-482

(Version 3) 5736-CB2 (Compiler Only)

5736-LM2 (Library Only)

5736-CB3 (Compiler and Library)

This publication gives the programmer the rules for writing programs that are to be compiled by the IBM Full American National Standard COBOL compilers under the Disk Operating System. It is meant to be used as a reference manual in the writing of IBM Full American National Standard COBOL programs.

COBOL (COmmon Business Oriented Language) is a programming language, similar to English, that is used for commercial data processing. It was developed by the Conference On DAta SYstems Languages (CODASYL). The standard of the language is American National Standard COBOL, X3.23-1968, as approved by the American National Standards Institute (ANSI).

IBM DOS Full American National Standard COBOL, Version 3, which includes all the features of earlier versions, incorporates the eight processing modules defined in the highest level of the American national standard. These modules include:

Nucleus
Table Handling
Sequential Access
Random Access
Sort
Report Writer
Segmentation
Library

A significant number of IBM extensions are implemented as well; these extensions are printed on a shaded background. This IBM implementation of American National Standard COBOL also complies with the first Draft ISO recommendation on COBOL.

This publication describes the IBM implementation of Full American National Standard COBOL, and all IBM extensions to that standard. Some statements are extensions to either American National Standard COBOL or to both American National Standard COBOL and the complete definition of CODASYL COBOL.

In this publication, the term standard COBOL means American National Standard COBOL; the term IBM Full American National Standard COBOL means this IBM implementation of American National Standard COBOL and all extensions to that standard. There are two types of extensions:

1. Those that represent features not specified by American National Standard COBOL.

2. Those that represent an easing of the strict American National Standard COBOL rules and allow for greater programming convenience.

All such extensions are printed on a shaded background for the convenience of users who wish strict conformance with the standard. Use of features that are extensions to the standard may result in incompatibilities between the implementation represented by this document and other implementations. If a complete chapter is an extension, only the chapter heading is shaded. These chapters are:

Subprogram Linkage Statements

Debugging Language

Format Control of the Source Program Listing

Sterling Currency

For the less experienced programmer, the introduction summarizes the general principles of COBOL, highlights features of American National Standard COBOL, and, through an example, illustrates the logical sequence and interrelationship of commonly used elements of a COBOL program. The balance of the publication gives the specific rules for correct programming in IBM Full American National Standard COBOL,

as implemented by the System/360 Disk Operating System. Appendixes provide supplemental information useful in writing COBOL programs. Appendix A describes the use of intermediate results in arithmetic operations; Appendix B contains several sample programs showing the use of mass storage files; Appendix C lists all of the formats and reserved words in IBM Full American National Standard COBOL; Appendix D is a file processing summary; Appendix E gives considerations for the use of ASCII encoded files; Appendix F explains the symbolic debugging feature; Appendix G explains 3525 combined function processing.

Compiler output and restrictions, programming examples, and information on running an IBM American National Standard COBOL program are found in the publication IBM DOS Full American National Standard COBOL Programmer's Guide, Order No. GC28-6398 and in the Program Product publication IBM DOS Full American National Standard COBOL Compiler and Library, Version 3, Programmer's Guide, Order No. SC28-6441. These programmer's guides and this language reference manual are corequisite publications.

A knowledge of basic data processing techniques is mandatory for the understanding of this publication. Such information, as it applies to System/360, can be found in the following publications:

Introduction to IBM Data Processing Systems, Form GC20-1684

Introduction to IBM System/360 Direct Access Storage Devices and Organization Methods, Form GC20-1649

The reader should also have a general knowledge of COBOL before using this manual. Useful background information can be found in the following publications:

COBOL Program Fundamentals: Text, Form R29-0205

COBOL Program Fundamentals: Reference Handbook, Form R29-0206

Writing Programs in COBOL: Text, Form R29-0210

Writing Programs in COBOL: Reference Handbook, Form R29-0211

COBOL Programming Techniques: Text, Form R29-0215

Where information in the foregoing publications conflicts with information in this publication, the contents herein supersede any other in the writing of COBOL programs. Any violation of the rules defined in this publication for using the Disk Operating System is considered an error.

A general knowledge of the IBM Disk Operating System is desirable, although not mandatory. The following publication gives such information:

IBM System/360 Disk and Tape Operating System: Concepts and Facilities, Form GC24-5030.

Tables

FEATURES OF THE PROGRAM PRODUCT VERSION 3 COMPILER

The IBM Disk Operating System Full American National Standard COBOL
Compiler Version 3 includes the following features:

- Improvements in Object Code to save main storage:

    (1) Optimized Object Code -- which results, when specified, in
         up to 30% space saving in object program generated code
         and global tables as compared with Version 2. The space
         saved depends on the number of referenced procedure-names
         and branches, and on 01-level data names.

    (2) System/370 Support can be requested, to take advantage of
         the System/370 instruction set. When such support is
         requested, System/370 instructions particulary suited to
         COBOL programming are generated to replace the equivalent
         object-time subroutines and instructions needed when
         running under System/360. The System/370 instructions
         save up to 12% of generated object program space, plus the
         space no longer needed by the subroutines.

    (3) Double-Buffered ISAM -- allows faster sequential
         processing of indexed files.

    (4) Improvements in the MOVE Statement and in Comparisons --
         when a MOVE statement or a comparison involves a one-byte
         literal, generated code for the move and the comparison
         has been improved. This saves object program space.

    (5) Improved DISPLAY Routines -- the DISPLAY routine has been
         split into subsets for more efficient object program code.

- Alphabetized Cross-Reference Listing (SXREF) -- for easier
  reference to user-specified names in a program. SXREF performs
  up to 25 times faster than previous source-ordered
  cross-reference (XREF). Version 3 XREF performance is improved
  by at least the same amount. The larger the source program, the
  more that performance is improved. Total compilation time is up
  to 3 times faster.

- Debugging Facilities that are more powerful and flexible

    (1) Symbolic Debug Feature -- which provides a symbolic
         formatted dump at abnormal termination, or a dynamic dump
         during program execution.

    (2) Flow Trace Option -- a formatted trace can be requested
         for a variable number of procedures executed before
         abnormal termination.

    (3) Statement Number Option -- provides information about the
         COBOL statement being executed at abnormal termination.

    (4) Expanded CLIST and SYM -- for more detailed information
         about the Data Division and Procedure Division.

    (5) Relocation Factor -- can be requested to be included in
         addresses on the object code listing for easier debugging.

    (6) Working-Storage Location and Size -- When CLIST and SYM
         are in effect, the starting address and size of
         Working-Storage are printed.

- <u>Additional Device Support</u> -- the following devices can be specified:

  3211 -- 150-character printer

  2319, 3330 -- mass storage (direct access) facilities

  3410, 3420 -- tape utility devices

  3505, 3525 -- advanced unit-record devices

- <u>ASCII Support</u> -- allows creation and retrieval of tape files written in the American National Standard Code for Information Interchange (ASCII).

- <u>Separately Signed Numeric Data Type</u> -- for more flexible numeric data description. The sign can be a separate character or an overpunch, and can be leading or trailing.

- <u>ON Statement Improvement</u> -- Count-conditional operands can be identifiers, allowing the programmer to change the conditions of execution of the statement at object time.

In 1959, a group of computer professionals, representing the U.S. Government, manufacturers, universities, and users, formed the Conference On DAta SYstems Language (CODASYL). At the first meeting, the conference agreed upon the development of a common language for the programming of commercial problems. The proposed language would be capable of continuous change and development, it would be problem-oriented and machine-independent, and it would use a syntax closely resembling English, avoiding the use of special symbols as much as possible. The COmmon Business Oriented Language (COBOL) which resulted met most of these requirements.

As its name implies, COBOL is especially efficient in the processing of business problems. Such problems involve relatively little algebraic or logical processing; instead, they usually manipulate large files of similar records in a relatively simple way. This means that COBOL emphasizes the description and handling of data items and input/output records.

In the years since 1959, COBOL has undergone considerable refinement and standardization, and a standard COBOL has been approved by ANSI (American National Standards Institute), an industry-wide association of computer manufacturers and users; this standard is called American National Standard COBOL, X3.23-1968.

This publication explains IBM Full American National Standard COBOL, which is compatible with the highest level of American National Standard COBOL and includes a number of IBM extensions to it as well. The compiler supports the processing modules defined in the standard. These processing modules include:

NUCLEUS -- which defines the permissible character set and the basic elements of the language contained in each of the four COBOL divisions: Identification Division, Environment Division, Data Division, and Procedure Division.

TABLE HANDLING -- which allows the definition of tables and making reference to them through subscripts and indexes. A convenient method for searching a table is provided.

SEQUENTIAL ACCESS -- which allows the records of a file to be read or written in a serial manner. The order of reference is implicitly determined by the position of the logical record in the file.

RANDOM ACCESS -- which allows the records of a file to be read or written in a manner specified by the programmer. Specifically defined keys, supplied by the programmer, control successive references to the file.

SORT -- which provides the capability of sorting files in ascending and/or descending order. This feature also includes procedures for handling such files both before and after they have been sorted.

REPORT WRITER -- which allows the programmer to describe the format of a report in the DATA DIVISION, thereby minimizing the amount of PROCEDURE DIVISION coding necessary.

SEGMENTATION -- which allows large problem programs to be split into segments that can then be designated as permanent or overlayable core storage. This assures more efficient use of core storage at object time.

LIBRARY -- which supports the retrieval and updating of pre-written source program entries from a user's library, for inclusion in a COBOL program at compile time. The effect of the compilation of library text is as though the text were actually written as part of the source program.

In this publication, the features included in the NUCLEUS, SEQUENTIAL ACCESS, and RANDOM ACCESS modules are presented as part of the discussion of "Language Considerations" and of the four divisions of a COBOL program. The other five modules -- TABLE HANDLING, SORT, REPORT WRITER, LIBRARY, and SEGMENTATION -- are presented as separate features of USA Standard COBOL.

This manual describes all versions of IBM System/360 Disk Operating System Full American National Standard COBOL. All information relating to the Program Product Version 3 compiler is presented within separate paragraphs. Such paragraphs begin with the heading "Program Product Information -- Version 3," and all following paragraphs pertaining to such information are indented.

This chapter gives the reader a general understanding of the principles of IBM Full American National Standard COBOL (hereinafter simply termed "COBOL"). It introduces the reader to COBOL and demonstrates some of the ways in which the language can be used in the solution of commercial problems. This discussion does not define the rules for using COBOL, but rather attempts to explain the basic concepts of the language through relatively simple examples.

The reader who has an understanding of the principles of currently implemented versions of COBOL may wish to go directly to "Language Considerations." Other readers will find many concepts discussed in this chapter of help in using the detailed instructions throughout the rest of this manual.

PRINCIPLES OF COBOL

COBOL is one of a group of high-level computer languages. Such languages are problem oriented and relatively machine independent, freeing the programmer from many of the machine oriented restrictions of assembler language, and allowing him to concentrate instead upon the logical aspects of his problem.

COBOL looks and reads much like ordinary business English. The programmer can use English words and conventional arithmetic symbols to direct and control the complicated operations of the computer. The following are typical COBOL sentences:

        ADD DIVIDENDS TO INCOME.
        MULTIPLY UNIT-PRICE BY STOCK-ON-HAND
            GIVING STOCK-VALUE.
        IF STOCK-ON-HAND IS LESS THAN ORDER-POINT
            MOVE ITEM-CODE TO REORDER-CODE.

Such COBOL sentences are easily understandable, but they must be translated into machine language -- the internal instruction codes -- before they can actually be used.

A special systems program, known as a compiler, is first entered into the computer. The COBOL program (referred to as the source program) is then entered into the machine, where the compiler reads it and analyzes it. The COBOL language contains a basic set of reserved words and symbols. Each combination of reserved words and symbols is transformed

by the compiler into a definite set of usable machine instructions.   In
effect, the programmer has at his disposal a whole series of
"prefabricated" portions of the machine-language program he wishes the
compiler to construct.

When he writes a COBOL program, he is actually directing the compiler
to bring together, in the proper sequence, the groups of machine
instructions necessary to accomplish the desired result.   From the
programmer's instructions, the compiler creates a new program in machine
language.   This program is known as an object program.
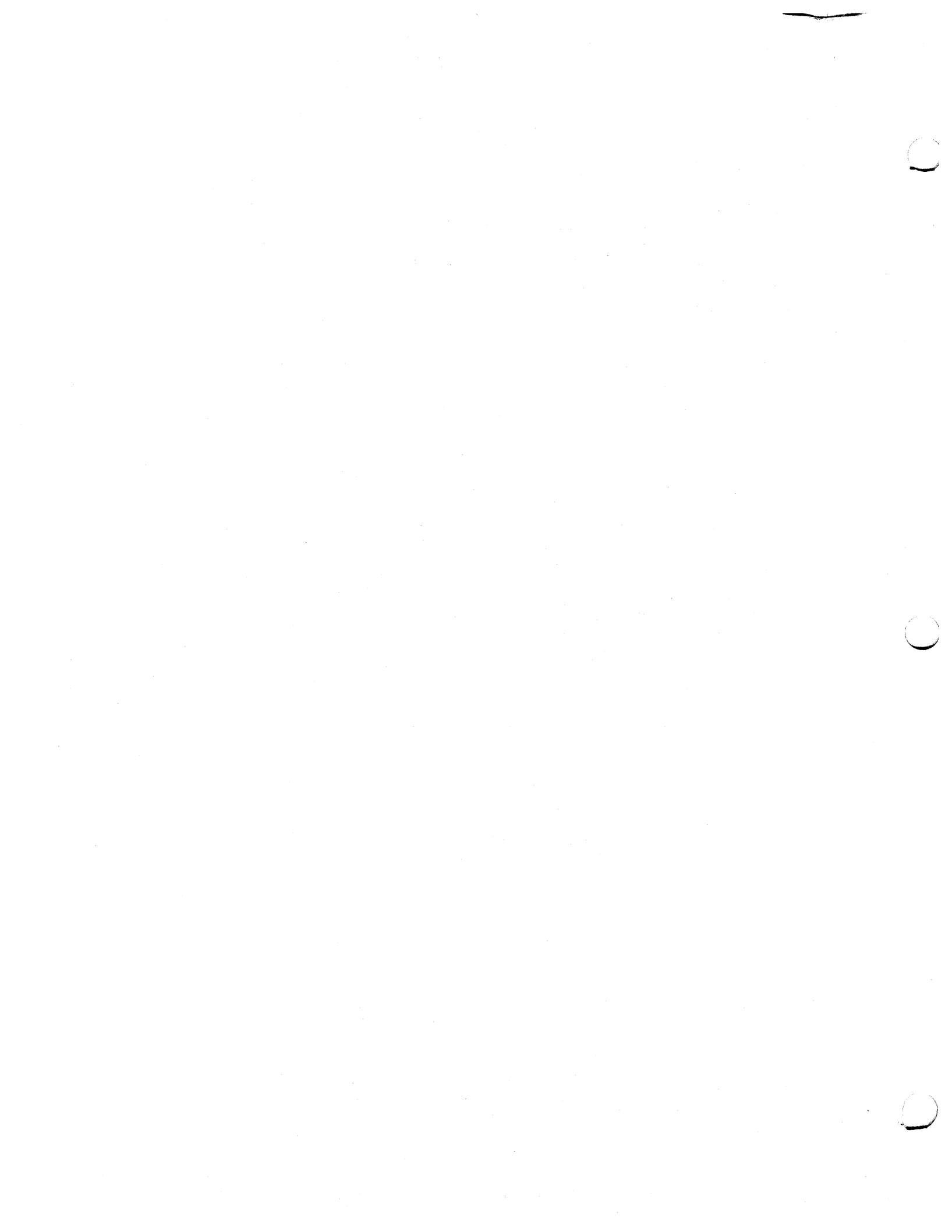
Once the object program has been produced, it may be used at once, or it may be recorded on some external medium and stored for future use. When it is needed, it can then be called upon again and again to process data.

Every COBOL program is processed first when the compiler translates the COBOL program into machine language (compile time), then when the machine language program actually processes the data (execution time).

A simple example illustrates the basic principles of translating a COBOL sentence. To increase the value of an item named INCOME by the value of an item named DIVIDENDS, the COBOL programmer writes the following sentence:

ADD DIVIDENDS TO INCOME.

Before the compiler can interpret this sentence, it must be given certain information. The programmer describes the data represented by the names DIVIDENDS and INCOME in such a way that the compiler can recognize it, obtain it when needed, and treat it in accordance with its special characteristics.

First, the compiler examines the word ADD. It determines whether or not ADD is one of the COBOL reserved words, that is, words that have clearly defined meanings in COBOL (rather than a word like DIVIDENDS, which is defined by the programmer). ADD is a special kind of reserved word--a COBOL key word. Therefore, the compiler generates the machine instructions necessary to perform an addition and inserts them into the object program.

The compiler next examines the word DIVIDENDS. Because the programmer has supplied data information about DIVIDENDS, the compiler knows where and how DIVIDENDS information is to be placed in core storage, and it inserts into the object program the instructions needed in order to locate and obtain the data.

When the compiler encounters the word TO, it again determines whether or not this is a COBOL reserved word. It is such a word, and the compiler interprets it to mean that the value represented by the name following the word TO, in this case INCOME, must be increased as a result of the addition.

The compiler next examines the word INCOME. Again, it has access to data information about the word. As a result, it is able to place in the object program the instructions necessary to locate and use INCOME data.

The programmer placed a period after the word INCOME. The effect of the period on the COBOL compiler is similar to its effect in the English language. The period tells the compiler that it has reached the last word to which the verb ADD applies, the end of the sentence.

The logical steps we have described are performed by the compiler in creating the object program, although they might not be performed in exactly this sequence. All these preparatory steps are required only in creating the object program. Once created, the object program is used for the actual processing and may be saved for future reference. The source program is not required further, unless the programmer makes a change in it; in that case, it must be compiled again to create a new object program.

When the machine-language instruction for ADD is actually performed at execution time, the instruction is executed in either of two ways, depending on the format of the data:

1. It directly adds the value of DIVIDENDS to the value of the data representing INCOME, thus giving the new value of INCOME.

or

2. It moves the data representing INCOME into a special work area, or register; then DIVIDENDS is added to it to create the sum, after which the new value of INCOME is returned to the proper area in storage.

In this simple example, the object program could add the two specified items with very few machine instructions. In actual practice, however, some complex COBOL sentences produce dozens of machine instructions. Then, too, a computer can be instructed to repeat a procedure any number of times. A few COBOL sentences can start the computer on operations that could process millions of data records rapidly and accurately.

A SAMPLE COBOL PROGRAM

COBOL is based on English; it uses English words and certain syntax rules derived from English. However, because it is a computer language, it is much more precise than English. The programmer must, therefore, learn the rules that govern COBOL and follow them exactly. These rules are detailed later, beginning in the next chapter. The rest of this chapter gives a general picture of how a COBOL program is put together.

The basic unit of COBOL is the word -- which may be a COBOL reserved word or a programmer-defined word. Reserved words have a specific syntactical meaning to the COBOL compiler, and must be spelled exactly as shown in the reserved word list (see Appendix C). Programmer-defined words are assigned by the user to such items as data-names and procedure-names; they must conform to the COBOL rules for the formation of names.

Reserved words and programmer-defined words are combined by the programmer into clauses (in the Environment and Data Divisions) and statements (in the Procedure Division); clauses and statements must be formed following the specific syntactical rules of COBOL. A clause or a statement specifies only one action to be performed, one condition to be analyzed, or one description of data. Clauses and statements can be combined into sentences. Sentences may be simple (one statement or one clause), or they may be compound (a combination of statements or a combination of clauses). Sentences can be combined into paragraphs, which are named units of logically related sentences, and paragraphs can be further combined into named sections. Both paragraphs and sections can be referred to as procedures, and their names can be referred to as procedure names. Procedures (sections and paragraphs) are combined into divisions.

There are four divisions in each COBOL program. Each is placed in its logical sequence, each has its necessary logical function in the program, and each uses information developed in the divisions preceding it. The four divisions and their sequence are:

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION.
```

To illustrate how a COBOL program is written, let us create a simplified procedure to record changes in the stocks of office furniture offered for sale by a manufacturer. We will need such data items as an item code to identify each type of product, an item name corresponding to the code, the unit price of each item of stock, the reorder point at

which the manufacturer replaces each item, and the amount of stock on hand plus its value for each item. Our procedure will update a MASTER-FILE of all stocks the manufacturer carries by reading a DETAIL-FILE of current transactions, performing the necessary calculations, and placing the updated values in the MASTER-FILE. We will also create an ACTION-FILE of items to be reordered. The MASTER-FILE resides on a direct access (mass storage) disk device; the DETAIL-FILE and ACTION-FILE reside on tape devices.

Many of the examples used in the following discussion have been simplified for greater clarity. Figure 2, at the end of this chapter, shows how the entire UPDATING program would actually be written.

Identification Division

First we must assign a name to our program, presenting the information like this:

    IDENTIFICATION DIVISION.
    PROGRAM-ID. UPDATING.

PROGRAM-ID informs the compiler that we have chosen the unique name UPDATING for the program we have written.

In addition to the name of the program, the Identification Division allows us to list the name of the programmer, the date the program was written, and other information that will serve to document the program.

Environment Division

Although COBOL is, to a large degree, machine independent, there are some aspects of any program that depend on the particular computer being used and on its associated input/output devices. In the Environment Division, the characteristics of the computer used may be identified. The location of each file referenced in the program, and how each one of them will be used, must be described.

First we will describe the source computer (the one the compiler uses) and the object computer (the one the object program uses) as follows:

    ENVIRONMENT DIVISION.
    CONFIGURATION SECTION.
    SOURCE-COMPUTER.   IBM-360-F50.
    OBJECT-COMPUTER.   IBM-360-F50.

This tells us that both computers will be an IBM System/360 model F50.

Next we must identify the files to be used in our program, and assign them to specific input/output devices. This is done in the Input-Output Section.

```
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT MASTER-FILE, ASSIGN TO ...
        ACCESS MODE IS RANDOM
        ACTUAL KEY IS FILEKEY.
    SELECT DETAIL-FILE, ASSIGN TO ...
        ACCESS MODE IS SEQUENTIAL.
    SELECT ACTION-FILE, ASSIGN TO ...
        .
        .
        .
```

The ellipses (...) in the three foregoing ASSIGN clauses indicate the omission of system-name, an item too complex to illustrate here. System-name is in a special format, and it tells the compiler on which symbolic unit the file will be found, on what kind of device the file resides, and in what way the data is organized within the file.

Our MASTER-FILE resides on a disk pack, which is a mass storage device. Access for these devices can be either RANDOM or SEQUENTIAL. If ACCESS MODE IS RANDOM, then each record within the file can be located directly through the use of a key (identified in the statement ACTUAL KEY IS FILEKEY). For our program we have named this key FILEKEY, and later in the Data Division we will describe it fully. During the processing of our object program, each record will be made available to the user in the sequence that the keys are presented to the system.

Our DETAIL-FILE and our ACTION-FILE reside on tape. This means that ACCESS MODE must be sequential. On tape it is necessary to refer to each successive record in the file in order to find any individual record we might wish to access. Since the compiler assumes that the ACCESS MODE is sequential unless specified otherwise, the ACCESS MODE clause is never needed in describing a tape file.

Data Division

The Data Division of the COBOL program gives a detailed description of all the data to be used in the program -- whether to be read into the machine, used in intermediate processing, or written as output. To simplify this discussion, we will describe only the two most important aspects of data description.

1. We will inform the compiler that we intend to work with one kind of input record, our detail record; one kind of update record, our master record; and one kind of output record, our action record.

2. We will assign data-names to each of the items of data to be used.

First, we must organize the two input records -- a MASTER-RECORD and a DETAIL-RECORD. The MASTER-RECORD will be derived from ledger records that look like those shown in Table 1.

Table 1. Typical Ledger Records Used for MASTER-RECORD

Item Code	Item Name	Stock on Hand	Unit Price ($)	Stock Value ($)	Order Point
A10	2-drawer file cabinets	100	50	5,000	50
A11	3-drawer file cabinets	175	80	14,000	80
A12	4-drawer file cabinets	200	110	22,000	150
B10	Secretarial desks	150	200	30,000	120
B11	Salesmen's desks	50	175	8,750	50
B12	Executive desks	75	500	37,500	60
C10	Secretarial posture chairs	125	50	6,250	140
C11	Side chairs	50	40	2,000	60
C12	Executive swivel chairs	25	150	3,750	20

There will be a MASTER-RECORD for each item in this list. In defining the data for the compiler, we will make sure that each record is in the same format as all the others. Thus, if we specify the characteristics of a single record, we will have specified the characteristics of the whole set. In this way, all of the master records can be organized into a data set, or file, that we will name MASTER-FILE. Each complete record within the file we will name the MASTER-RECORD, with the individual items of data grouped within it. Accordingly, we will begin our Data Division as follows:

```
DATA DIVISION.
FILE SECTION.
FD   MASTER-FILE DATA RECORD IS MASTER-RECORD...
       .
       .
       .
01   MASTER-RECORD.
       02   ITEM-CODE...
       02   ITEM-NAME...
       02   STOCK-ON-HAND...
       02   UNIT-PRICE...
       02   STOCK-VALUE...
       02   ORDER-POINT...
              .
              .
              .
```

The FILE SECTION entry informs the COBOL compiler that the items that follow will describe the format of each file and of each record within each file to be used in the program. The level indicator FD (File Description) introduces the MASTER-FILE itself, and tells the compiler that each entry within MASTER-FILE will be referred to as MASTER-RECORD. The entry with level number 01 identifies the MASTER-RECORD itself, and the subordinate entries with level number 02 describe the subdivisions within the complete MASTER-RECORD. The concept of levels is a basic attribute of COBOL. The highest level is the FD, the next highest level is 01. Level numbers from 02 through 49 may subdivide the record, and the subdivisions themselves can be further subdivided if need be. The smaller the subdivision, the larger the level number must be.

Each of the data items would actually be described more fully than is shown here. In an actual program, for example, we would inform the compiler that each of the items identified as STOCK-ON-HAND, UNIT-PRICE, STOCK-VALUE, and ORDER-POINT would represent postive numeric values of a specific size in a specific form, and so forth. At this point, we need not concern ourselves with these details.

The MASTER-FILE is the main record of current inventory. Changes to this record are made by entering the details of individual transactions or groups of transactions. Thus, receipts of new stocks and shipments to customers will change both STOCK-ON-HAND and STOCK-VALUE. These changes are summarized in the detail record for each item. A typical record would appear in a ledger as shown in Table 2.

Table 2. Typical DETAIL-RECORD

Item Code	Item Name	Receipts	Shipments
B11	Salesmen's desks	25	15

We will therefore organize a DETAIL-FILE, made up of individual items to be referred to as DETAIL-RECORD. DETAIL-FILE will be arranged by ITEM-CODE in ascending numerical order.

```
FD   DETAIL-FILE DATA RECORD IS DETAIL-RECORD...
01   DETAIL-RECORD.
     02   ITEM-CODE...
     02   ITEM-NAME...
     02   RECEIPTS...
     02   SHIPMENTS...
```

The ACTION-FILE will contain a list of items to be reordered, plus relevant data:

```
FD   ACTION-FILE DATA RECORD IS ACTION-RECORD...
01   ACTION-RECORD.
     02   ITEM-CODE...
     02   ITEM-NAME...
     02   STOCK-ON-HAND...
     02   UNIT-PRICE...
     02   ORDER-POINT...
```

This completes the description of the files we will use.

Note that the names of data items contained within the files are in many cases identical. Yet each name within each file must be unique, or ambiguities in references to them will occur. Since identical names are used in our data descriptions, we must use a special means of distinguishing between them. The COBOL naming system, with its concept of levels, allows us to make this distinction by reference to some larger group of data of which the item is a part. Thus, ITEM-CODE OF MASTER-RECORD, and ITEM-CODE OF DETAIL-RECORD, and ITEM-CODE OF ACTION-RECORD can be clearly differentiated from each other. The use of a higher level name in this way is called qualification. Qualification is required in making distinctions between otherwise identical names.

Now we must construct the Working-Storage Section of our Data Division. This section describes records and data items that are not part of the files, but are used during the processing of the object program.

For our program, we will need several entries in our Working-Storage Section. Among them will be several items constructed with level numbers, similar to those used to describe the file records.

```
WORKING-STORAGE SECTION.
            •
            •
            •
    77   QUOTIENT...
            •
            •
            •
    01   THE-KEY...
         02   FILLER...
         02   FILEKEY...
    01   ERROR-MESSAGE.
         02   ERROR-MESSAGE-1...
         02   ERROR-MESSAGE-2...
         02   ERROR-MESSAGE-3...
```

We will use THE-KEY record in constructing the FILEKEY.   The
ERROR-MESSAGE record we will use to create warning messages when errors
are encountered during object time processing.   The data item named
QUOTIENT we have assigned the level number 77.   This level number
informs the compiler that QUOTIENT is a noncontiguous data item -- that
is, that this item has no relationship to any other data item described
in the Working-Storage Section.   Note that the data items related to
each other must be listed after all the noncontiguous data items.

Procedure Division

The Procedure Division contains the instructions needed to solve our
problem.   To accomplish this, we will use several types of COBOL
statements.   In constructing our sample program, we will discover how
each type of statement can be used to obtain the results we want.

Beginning the Program -- Input Operations

Our first step in building the Procedure Division is to make the
records contained in the MASTER-FILE and the DETAIL-FILE available for
processing.   If we write the statements:

```
PROCEDURE DIVISION.
            •
            •
            •
    OPEN INPUT DETAIL-FILE.
    OPEN I-O MASTER-FILE.
```

the system establishes a line of communication with each file, checks to
make sure that each is available for use, brings the first record of the
DETAIL-FILE file into special areas of internal storage known as
buffers, and does other housekeeping.

The files can now be accessed. Our next statements will therefore be:

        READ DETAIL-FILE AT END GO TO END-ROUTINE.
                •
                •
                •
        READ MASTER-FILE INVALID KEY PERFORM INPUT-ERROR
        GO TO ERROR-ROUTINE-1.

At this point in our program, these two statements make available for processing the first record from each file. (Note that the AT END phrase and the INVALID KEY phrase are necessary in these sentences. Their use will be explained later.) We are now able to begin arithmetic operations upon the data.

Arithmetic Statements

We have already seen that the COBOL language contains the verb ADD. Using this verb, we can add RECEIPTS to STOCK-ON-HAND by writing the COBOL statement:

    ADD RECEIPTS TO STOCK-ON-HAND.

This instructs the program to find the value of RECEIPTS in the DETAIL-RECORD and add it to the value of STOCK-ON-HAND in the MASTER-RECORD. (For the sake of brevity, this example and the ones following have been simplified by omitting the name qualification which would be necessary in actual coding. Figure 2, at the end of this chapter, shows the actual coding necessary.)

Next we must reduce the new value of STOCK-ON-HAND by the amount of SHIPMENTS. The COBOL verb SUBTRACT will accomplish this result for us, and so we write:

    SUBTRACT SHIPMENTS FROM STOCK-ON-HAND.

These two statements, carried out in succession, will produce a current value for STOCK-ON-HAND.

Actually, there is a more concise way to perform this particular calculation. We have broken it into two steps, but COBOL provides another verb which allows us to specify more than one arithmetic operation in a single statement. This is the verb COMPUTE.

    COMPUTE STOCK-ON-HAND = STOCK-ON-HAND + RECEIPTS - SHIPMENTS.

A COMPUTE statement is always interpreted to mean that the value on the left of the equal sign will be changed to equal the value resulting from the calculation specified on the right. The calculation on the right of the equal sign is evaluated from left to right. That is, in our example, the addition is performed first and then the subtraction.

The name STOCK-ON-HAND occurs twice in this sentence, but this causes no difficulty. The expression to the right is calculated first; thus, it is the current value of STOCK-ON-HAND that is used as the basis for computing the new value. When this new value has been calculated, it replaces the old value of STOCK-ON-HAND in the MASTER-RECORD.

So far we have brought only the value of STOCK-ON-HAND up to date, but a change in this value will also cause a change in STOCK-VALUE. We will assume that this figure does not include allowances for quantity discounts, damage to stock, or other such factors, and that STOCK-VALUE is nothing more than the unit price multiplied by the number of items currently in stock. COBOL provides us with a MULTIPLY verb, which permits us to accomplish this:

```
MULTIPLY STOCK-ON-HAND BY UNIT-PRICE GIVING STOCK-VALUE.
```

The result of the multiplication will be placed in the MASTER-RECORD as the new value of STOCK-VALUE. Within the program, this statement must be executed after the COMPUTE statement we wrote earlier, since STOCK-ON-HAND must be the updated, not the original, value.

Conditional Statements

There are instructions in COBOL that examine data to determine whether or not some condition is present and, depending on what is found, to carry out an appropriate course of action.

The MASTER-RECORD contains an item called ORDER-POINT. An item is to be reordered when its stock has been reduced either to or below its order point. Let us assume that we have written a procedure for initiating such an order, and that we have given the name REORDER-ROUTINE to this procedure. We then write the following two sentences:

```
IF STOCK-ON-HAND IS LESS THAN ORDER-POINT
    PERFORM REORDER-1...
IF STOCK-ON-HAND IS EQUAL TO ORDER-POINT
    PERFORM REORDER-1...
```

in order to compare the present value of STOCK-ON-HAND with the value of ORDER-POINT. If STOCK-ON-HAND is a smaller value, the COBOL verb PERFORM causes a transfer of control to the paragraph named REORDER-1. If STOCK-ON-HAND is not less than ORDER-POINT, our next instruction is evaluated. If the values are equal, control is transferred to REORDER-1. If the values are not equal, control is transferred to the next instruction.

It is permissible, in COBOL, to combine the two tests into one:

```
IF STOCK-ON-HAND IS LESS THAN ORDER-POINT OR EQUAL TO
    ORDER-POINT PERFORM REORDER-1...
```

Here we are writing a compound condition with an implied subject. STOCK-ON-HAND, the subject of the first condition, is understood to be the subject of the second condition as well. Compound conditions increase the flexibility of COBOL and make the handling of many kinds of problems easier.

In this example, we tested successively for two conditions out of three. Unless the programmer has some need to distinguish between these two conditions (and he might), it would be simpler to test for the third condition instead:

```
IF STOCK-ON-HAND IS GREATER THAN ORDER-POINT NEXT SENTENCE
    ELSE PERFORM REORDER-1...
```

The words NEXT SENTENCE have a special meaning in COBOL. When IF
STOCK-ON-HAND IS GREATER THAN ORDER-POINT is true, NEXT SENTENCE takes
effect. Every instruction in the balance of the IF sentence is ignored,
and control is transferred to the sentence following.

    The test can be simplified even further, since COBOL allows us to
express negation:

       IF STOCK-ON-HAND IS NOT GREATER THAN ORDER-POINT
           PERFORM REORDER-1...

If the value of STOCK-VALUE is less than or equal to that of
ORDER-POINT, control is transferred to REORDER-1. If the value is
greater, control automatically passes to the next successive sentence.

    The actual rules for specifying tests and comparisons will be given
in a subsequent chapter.

Handling Possible Errors

    Let us write one more conditional statement:

      IF STOCK-ON-HAND IS LESS THAN ZERO...
           GO TO ERROR-WRITE.

One would expect that the smallest value STOCK-ON-HAND could assume
would be zero. If a negative record were processed, the values found
would probably be completely erroneous. To prevent this, the programmer
could anticipate the possibility of error and write a special routine to
be executed whenever the value of STOCK-ON-HAND was found to be
negative. Such a routine could stop the processing of this record,
print out the erroneous data, and proceed automatically to process the
records following. The more comprehensive a programmer makes his error
checking, the less likely it is that inaccurate information will pass
through without being marked for special attention.

Data-Manipulation Statements

    We saw in the foregoing that if the value of STOCK-ON-HAND fell below
a certain point, control would be passed to a special sequence of
instructions named REORDER-1. Our output ACTION-FILE has been set up
for just this purpose. The bulk of REORDER-1 could consist of
data-manipulation statements; that is, instructions which move the
necessary data items from the MASTER-RECORD area in storage to that area
reserved for the ACTION-FILE records. The COBOL verb MOVE can be used
to accomplish this. We must explain here that the verb MOVE does not
mean an actual physical movement of data. Instead, it means that the
data items from MASTER-RECORD are copied into ACTION-RECORD. Items
within MASTER-RECORD are not destroyed when a MOVE statement is
executed, and are available for further processing. Individual items

contained in ACTION-RECORD before the operation, however, are replaced
when the statement is executed. Our MOVE statements will be written:

```
MOVE ITEM-CODE OF MASTER-RECORD TO ITEM-CODE
    OF ACTION-RECORD.
MOVE ITEM-NAME OF MASTER-RECORD TO ITEM-NAME
    OF ACTION-RECORD.
MOVE STOCK-ON-HAND OF MASTER-RECORD TO
    STOCK-ON-HAND OF ACTION-RECORD.
MOVE UNIT-PRICE OF MASTER-RECORD TO UNIT-PRICE
    OF ACTION-RECORD.
MOVE ORDER-POINT OF MASTER-RECORD TO ORDER-POINT
    OF ACTION-RECORD.
```

With these five statements, we have set up the ACTION-RECORD to be
written in the ACTION-FILE. However, there is another and easier method
for the programmer to specify the five MOVE operations by taking
advantage of the qualification system in naming:

```
MOVE CORRESPONDING MASTER-RECORD TO ACTION-RECORD.
```

The word CORRESPONDING indicates that those data items with names which
are identical in both records are to be copied from MASTER-RECORD into
ACTION-RECORD. Thus, five MOVE statements are replaced by one.

Output Operations

    When all arithmetic and data-manipulation statements have been
executed, we will write the results in some form. COBOL allows us to do
this with a WRITE instruction.

```
WRITE MASTER-RECORD INVALID KEY ...
    GO TO ERROR-WRITE.
```

Or, if we were to indicate that an item was to be reordered, we could
write the following:

```
WRITE ACTION-RECORD.
```

In either case, the record would be recorded on the output device
specified for the file in the Environment Division; its format would be
determined by the Data Division description of the file.

Procedure Branching Statements

In our inventory problem, there will be as many master records as there are kinds of furniture in stock, and there will be a varying number of detail records. We must read each successive DETAIL-RECORD in DETAIL-FILE, until every one of the records in the file has been processed.

Each time a DETAIL-RECORD is read, we will perform calculations upon its ITEM-CODE in order to produce our FILEKEY. FILEKEY will then be used to find a matching record in MASTER-RECORD. If a matching record cannot be found, either the DETAIL-RECORD is in error, or the MASTER-RECORD is missing from the file and we must mark that record for special processing. Consider the series of statements in Figure 1.

You will note that several new elements have been added to the arithmetic statements and conditional phrases we have already discussed. First, there are the elements that extend to the left of the other statements. These elements are the procedure-names we described earlier. Each procedure-name indicates the beginning of a paragraph or a section within the program, and each indicates a reference point for programmer-specified transfer of control. When a procedure is entered, each logically successive instruction is processed in turn.

The procedure-names give us a means of controlling the processing of successive items in our DETAIL-FILE. If, for example, we have finished processing one complete DETAIL-RECORD and wish to begin processing the next, control must be transferred to NEXT-DETAIL-RECORD-ROUTINE. This is accomplished through the use of the COBOL verb GO TO, which transfers control to the procedure indicated, as in the statement:

    GO TO NEXT-DETAIL-RECORD-ROUTINE.

Processing then continues with the first sentence following the procedure name NEXT-DETAIL-RECORD-ROUTINE. Note the many other examples of the GO TO statement in our program. Each gives us the means of transferring control from one procedure to another.

Another way in which to control the processing of a series of records is through the use of the COBOL verb PERFORM. Like the verb GO TO, the verb PERFORM specifies a transfer to the first sentence of a routine. In addition, PERFORM provides various ways of determining the manner in which the procedure is to be processed.

Within the COMPUTATION-ROUTINE, there is a statement which uses the COBOL verb PERFORM:

    IF STOCK-ON-HAND IN MASTER-RECORD IS LESS THAN ZERO
        PERFORM DATA-ERROR GO TO ERROR-WRITE.

When STOCK-ON-HAND is computed to be less than zero, an error condition has occurred. First, the compiler is instructed to transfer control to a procedure named DATA-ERROR. Within DATA-ERROR, there is a MOVE statement which copies the characters within quotation marks ("DATA ERROR ON INPUT ") into the area of storage reserved for ERROR-MESSAGE-1. (The characters within quotation marks are what is known as a literal -- because they literally mean themselves. When ERROR-MESSAGE is displayed, these words will be an actual part of the error message.) Control is now transferred back to the next statement following the PERFORM statement, which is the GO TO ERROR-WRITE statement.

```
NEXT-DETAIL-RECORD-ROUTINE.
    READ DETAIL-FILE AT END GO TO END-ROUTINE-1.
            .
            .
            .
    READ MASTER-FILE INVALID KEY PERFORM INPUT-ERROR
        GO TO ERROR-WRITE.
COMPUTATION-ROUTINE.
            .
            .
            .
    IF STOCK-ON-HAND IN MASTER-RECORD IS LESS THAN ZERO
        PERFORM DATA-ERROR GO TO ERROR-WRITE.
            .
            .
            .
    IF STOCK-ON-HAND IN MASTER-RECORD IS NOT GREATER THAN
        ORDER-POINT IN MASTER-RECORD PERFORM REORDER-1
        THRU REORDER-2.
WRITE-MASTER-ROUTINE.
            .
            .
            .
    GO TO NEXT-DETAIL-RECORD-ROUTINE.
REORDER-1.
    GO TO SWITCH-ROUTINE.
SWITCH-ROUTINE.
    ALTER REORDER-1 TO   REORDER-2
        END-ROUTINE-1 TO   END-ROUTINE-3.
    OPEN OUTPUT ACTION-FILE.
REORDER-2.
    MOVE CORRESPONDING MASTER-RECORD TO ACTION-RECORD.
    WRITE ACTION-RECORD.
ERROR-WRITE.
            .
            .
            .
    GO TO NEXT-DETAIL-RECORD-ROUTINE.
INPUT-ERROR.
    MOVE " KEY ERROR ON INPUT " TO ERROR-MESSAGE-1.
            .
            .
            .
DATA-ERROR.
    MOVE "DATA ERROR ON INPUT " TO ERROR-MESSAGE-1.
            .
            .
            .
END-ROUTINE-1.
    GO TO END-ROUTINE-2.
END-ROUTINE-3.
    CLOSE ACTION-FILE.
END-ROUTINE-2.
    CLOSE DETAIL-FILE.
    CLOSE ACTION-FILE.
    STOP RUN.
```

Figure 1.   Illustration of Procedure Branching

Note that within COMPUTATION-ROUTINE there is another PERFORM statement that is processed in a similar manner:

```
IF STOCK-ON-HAND IN MASTER-RECORD IS NOT GREATER THAN
    ORDER-POINT IN MASTER-RECORD
    PERFORM REORDER-1 THRU REORDER-2.
```

This time, the PERFORM statement instructs the object program to process several paragraphs before returning control to the next successive statement. Thus, when this PERFORM statement is executed, control is transferred to REORDER-1. This paragraph is executed, the next paragraph, SWITCH-ROUTINE, is also executed, and then all the statements contained in REORDER-2 are executed, at which point control is returned to the first statement in WRITE-MASTER-ROUTINE -- the next successive statement after the PERFORM statement.

A PERFORM statement may specify that a single section or paragraph be processed, or, if the desired procedure consists of more than one section or paragraph, it can specify two names that identify the beginning and the end of the procedure.

GO TO and PERFORM statements may seem to do much the same job. Yet there are specific reasons that will cause the programmer to choose one over the other. On the one hand, the programmer may wish to transfer control to the same procedure from two entirely different sections of the program. In this case, PERFORM offers the most convenient method of returning to the point from which the transfer was made. On the other hand, if the programmer wishes to proceed to a portion of the program without specifying a return to the current routine, a GO TO statement will provide the best method of making the transfer.

In addition to the GO TO and PERFORM statements, there is another COBOL statement that affects procedure branching: the ALTER statement.

In any given execution of our object program, we may or may not use our ACTION-FILE. Only if some item in STOCK-ON-HAND has fallen below REORDER-POINT will it be necessary to create an ACTION-RECORD. Therefore, depending upon the data that is being processed, we will open ACTION-FILE only if and when such an operation is necessary.

Suppose that for the first time in a particular execution of our object program we have encountered a value for STOCK-ON-HAND that indicates it must be reordered. The statement:

```
IF STOCK-ON-HAND IN MASTER-RECORD IS NOT GREATER THAN
    ORDER-POINT IN MASTER-RECORD
    PERFORM REORDER-1 THRU REORDER-2.
```

instructs the compiler, when STOCK-ON-HAND is not greater than ORDER-POINT, to transfer control to the first sentence in REORDER-1. REORDER-1 consists of but one statement:

```
GO TO SWITCH-ROUTINE.
```

SWITCH-ROUTINE, as it happens, is the next paragraph, and it contains an ALTER statement:

```
ALTER REORDER-1 TO REORDER-2
    END-ROUTINE-1 TO END-ROUTINE-3.
```

This statement instructs the compiler to substitute the words REORDER-2 for SWITCH-ROUTINE (within REORDER-1), and END-ROUTINE-3 for END-ROUTINE-2 (within END-ROUTINE-1). Since, at the time the ALTER statement is executed, we are already beyond the point at which the

substitution is to be made in REORDER-1, we continue processing each sequential statement until we reach the end of REORDER-2. We open ACTION-FILE, and so forth, until we return control to the next statement following the PERFORM statement.

However, in this execution of our object program, the next time we must reorder an item, a different sequence of statements is performed. The program transfers control to REORDER-1, but now the GO TO statement within REORDER-1 has a different operand. Instead of SWITCH-ROUTINE, the program is now instructed to transfer control to the paragraph named REORDER-2. Through use of the ALTER statement, we have created a switch that bypasses the OPEN ACTION-FILE statement in subsequent processing of reordered items, since the OPEN statement need be executed but once in any execution of our object program.

Similarly, if ACTION-FILE was never opened in this execution of our object program, it is not necessary to close it. Therefore, the second part of the ALTER statement:

    END-ROUTINE-1 TO END-ROUTINE-3

allows alternate paths of program flow, depending on whether or not this ALTER statement was ever executed. The precise rules for programming the ALTER statement are given later in this publication; note, however, the increased programming flexibility it offers.

Ending the Program

One last step in the logic of our inventory program must now be taken. We have obtained the update information from a record, performed the needed arithmetic calculations, moved the data from one area of storage to another, and written the decision-making and procedure-branching instructions necessary to take care of special cases and to process each succeeding record. Then we have written the updated information into the MASTER-FILE, and, when necessary, have written the ACTION-FILE. We must now terminate the program after all records have been acted upon. Remember that we wrote our first READ statement as follows:

    READ DETAIL-FILE AT END GO TO END-ROUTINE-1

END-ROUTINE-1 will consist of the few instructions necessary to terminate operations for this program.

Just as the programmer made all the files available to the system with a set of OPEN instructions, he must now disconnect these same files with another series:

```
END-ROUTINE-1.
    GO TO END-ROUTINE-2.
END-ROUTINE-3.
    CLOSE ACTION-FILE.
END-ROUTINE-2.
    CLOSE DETAIL-FILE.
    CLOSE MASTER-FILE.
```

These instructions initiate necessary housekeeping routines. (Note here that, in our program, ACTION-FILE will be closed only if REORDER-1 THRU REORDER-2 has been performed and the ALTER statement has been executed.) Once a file has been closed, it cannot be accessed by the program again. The programmer now writes one last COBOL instruction, and it must be at the logical end of his processing:

STOP RUN.

At this point, COBOL ending procedures are initiated, and the execution of the program is halted.

This is only a general picture of the way in which a COBOL program works. The following chapters in this manual give detailed descriptions of all four divisions within a COBOL program, with explicit instructions for correct programming in IBM Full American National Standard COBOL.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. UPDATING.
REMARKS. THIS IS A SIMPLIFIED UPDATE PROGRAM, USED AS AN
    EXAMPLE OF BASIC COBOL TECHNIQUES. THE PROGRAM IS
    EXPLAINED IN DETAIL IN THE INTRODUCTION TO THIS MANUAL.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-360-F50.
OBJECT-COMPUTER. IBM-360-F50.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT MASTER-FILE ASSIGN TO SYS015-DA-2311-A-MASTER
        ACCESS MODE IS RANDOM
        ACTUAL KEY IS FILEKEY.
    SELECT DETAIL-FILE ASSIGN TO SYS007-UT-2400-S-INFILE
        ACCESS IS SEQUENTIAL.
    SELECT ACTION-FILE ASSIGN TO SYS008-UT-2400-S-OUTFILE.
DATA DIVISION.
FILE SECTION.
FD  MASTER-FILE LABEL RECORDS ARE STANDARD
    DATA RECORD IS MASTER-RECORD.
01  MASTER-RECORD.
    02  ITEM-CODE        PICTURE X(3).
    02  ITEM-NAME        PICTURE X(29).
    02  STOCK-ON-HAND    PICTURE S9(6)      USAGE COMP SYNC.
    02  UNIT-PRICE       PICTURE S999V99    USAGE COMP SYNC.
    02  STOCK-VALUE      PICTURE S9(9)V99   USAGE COMP SYNC.
    02  ORDER-POINT      PICTURE S9(3)      USAGE COMP SYNC.
FD  DETAIL-FILE LABEL RECORDS ARE OMITTED
    DATA RECORD IS DETAIL-RECORD.
01  DETAIL-RECORD.
    02  ITEM-CODE        PICTURE X(3).
    02  ITEM-NAME        PICTURE X(29).
    02  RECEIPTS         PICTURE S9(3)      USAGE COMP SYNC.
    02  SHIPMENTS        PICTURE S9(3)      USAGE COMP SYNC.
FD  ACTION-FILE LABEL RECORDS ARE OMITTED
    DATA RECORD IS ACTION-RECORD.
01  ACTION-RECORD.
    02  ITEM-CODE        PICTURE X(3).
    02  ITEM-NAME        PICTURE X(29).
    02  STOCK-ON-HAND    PICTURE S9(6)      USAGE COMP SYNC.
    02  UNIT-PRICE       PICTURE S999V99    USAGE COMP SYNC.
    02  ORDER-POINT      PICTURE S9(3)      USAGE COMP SYNC.
WORKING-STORAGE SECTION.
77  SAVE                 PICTURE S9(10)     USAGE COMP SYNC.
77  QUOTIENT             PICTURE S9999      USAGE COMP SYNC.
01  KEY-ACTUAL.
    02  M                PICTURE S999 COMP SYNC VALUE ZEROS.
    02  BB               PICTURE   S9 COMP SYNC VALUE ZEROS.
    02  CC               PICTURE S999 COMP SYNC VALUE ZEROS.
    02  HH               PICTURE   S9 COMP SYNC.
    02  R                PICTURE X VALUE LOW-VALUE.
    02  RECORD-ID        PICTURE X(29).
01  THE-KEY REDEFINES KEY-ACTUAL.
    02  FILLER           PICTURE X.
    02  FILEKEY          PICTURE X(37).
01  TRACK1               PICTURE 9(4).
01  TRACK2 REDEFINES TRACK1 COMP.
    02  CYL              PICTURE S999.
    02  HEAD             PICTURE   S9.
01  ERROR-MESSAGE.
    02  ERROR-MESSAGE-1 PICTURE X(20).
    02  ERROR-MESSAGE-2 PICTURE X(36).
    02  ERROR-MESSAGE-3 PICTURE X(46).
```

Figure 2. Complete UPDATING Program (Part 1 of 2)

```
PROCEDURE DIVISION.
OPEN-FILES-ROUTINE.
    OPEN INPUT DETAIL-FILE.
    OPEN I-O MASTER-FILE.
NEXT-DETAIL-RECORD-ROUTINE.
    READ DETAIL-FILE AT END GO TO END-ROUTINE-1.
NEXT-MASTER-RECORD-ROUTINE.
    MOVE ITEM-CODE IN DETAIL-RECORD TO SAVE.
    DIVIDE 19 INTO SAVE GIVING QUOTIENT
        REMAINDER TRACK1.
    ADD 1020 TO TRACK1.
    MOVE ITEM-NAME IN DETAIL-RECORD TO RECORD-ID.
    MOVE HEAD TO HH. MOVE CYL TO CC.
    READ MASTER-FILE INVALID KEY
        PERFORM INPUT-ERROR GO TO ERROR-WRITE.
COMPUTATION-ROUTINE.
    COMPUTE STOCK-ON-HAND IN MASTER-RECORD = STOCK-ON-HAND
        IN MASTER-RECORD + RECEIPTS - SHIPMENTS.
    IF STOCK-ON-HAND IN MASTER-RECORD IS LESS THAN ZERO
        PERFORM DATA-ERROR GO TO ERROR-WRITE.
    MULTIPLY STOCK-ON-HAND IN MASTER-RECORD BY UNIT-PRICE
        IN MASTER-RECORD GIVING STOCK-VALUE
        IN MASTER-RECORD.
    IF STOCK-ON-HAND IN MASTER-RECORD IS NOT GREATER THAN
        ORDER-POINT IN MASTER-RECORD PERFORM REORDER-1
        THRU REORDER-2.
WRITE-MASTER-ROUTINE.
    WRITE MASTER-RECORD INVALID KEY
        PERFORM OUTPUT-ERROR GO TO ERROR-WRITE.
    GO TO NEXT-DETAIL-RECORD-ROUTINE.
REORDER-1. GO TO SWITCH-ROUTINE.
SWITCH-ROUTINE.
    ALTER REORDER-1 TO  REORDER-2
        END-ROUTINE-1 TO  END-ROUTINE-3.
    DISPLAY "ACTION FILE UTILIZED".
    OPEN OUTPUT ACTION-FILE.
REORDER-2.
    MOVE CORRESPONDING MASTER-RECORD TO ACTION-RECORD.
    WRITE ACTION-RECORD.
ERROR-WRITE.
    MOVE DETAIL-RECORD TO ERROR-MESSAGE-2.
    DISPLAY ERROR-MESSAGE.
    GO TO NEXT-DETAIL-RECORD-ROUTINE.
INPUT-ERROR.
    MOVE " KEY ERROR ON INPUT " TO ERROR-MESSAGE-1.
    MOVE SPACES TO ERROR-MESSAGE-3.
DATA-ERROR.
    MOVE "DATA ERROR ON INPUT " TO ERROR-MESSAGE-1.
    MOVE MASTER-RECORD TO ERROR-MESSAGE-3.
OUTPUT-ERROR.
    MOVE "KEY ERROR ON OUTPUT " TO ERROR-MESSAGE-1.
    MOVE SPACES TO ERROR-MESSAGE-3.
END-ROUTINE-1.
    GO TO END-ROUTINE-2.
END-ROUTINE-3.
    CLOSE ACTION-FILE.
END-ROUTINE-2.
    CLOSE DETAIL-FILE.
    CLOSE MASTER-FILE.
    STOP RUN.
```

Figure 2. Complete UPDATING Program (Part 2 of 2)

- STRUCTURE OF THE LANGUAGE

- ORGANIZATION OF THE COBOL PROGRAM

- METHODS OF DATA REFERENCE

- USE OF THE COBOL CODING FORM

- FORMAT NOTATION

The COBOL language is so structured that the programmer can write his individual problem program within a framework of words that have particular meaning to the COBOL compiler. The result is the performance of a standard action on specific units of data. For example, in a COBOL statement such as MOVE NET-SALES TO CURRENT-MONTH, the words MOVE and TO indicate standard actions to the COBOL compiler. NET-SALES and CURRENT-MONTH are programmer-defined words which refer to particular units of data being processed by his problem program.

COBOL CHARACTER SET

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

Character	Meaning
0,1,...,9	digit
A,B,...,Z	letter
	space
+	plus sign
–	minus sign (hyphen)
*	asterisk
/	stroke (virgule, slash)
=	equal sign
$	currency sign
,	comma
;	semicolon
.	period (decimal point)
" or '	quotation mark
(	left parenthesis
)	right parenthesis
>	"greater than" symbol
<	"less than" symbol

Note: This compiler's default option for the quotation mark is the apostrophe ('). Unless the default option is overridden, the quotation mark (") may not be used. If conformance with the standard character set is desired, the programmer must specify the quotation mark (") through a CBL card at compile time. If the quotation mark is thus specified, the apostrophe (') may not be used.

Characters Used in Words

The characters used in words in a COBOL source program are the following:

    0 through 9
    A through Z
    – (hyphen)

A word is composed of a combination of not more than 30 characters chosen from the character set for words. The word cannot begin or end with a hyphen.

Character Set

Characters Used for Punctuation

The following characters are used for punctuation:

Character	Meaning
	space
,	comma
;	semicolon
.	period
" or '	quotation mark
(	left parenthesis
)	right parenthesis

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

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

2. A period, semicolon, or comma, when used, must not be preceded by a space, but must be followed by a space.

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

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

5. An arithmetic operator or an equal sign must always be preceded by a space and followed by a space. A unary operator may be preceded by a left parenthesis.

6. A comma may be used as a separator between successive operands of a statement. An operand of a statement is shown in a format as a lower-case word.

7. A comma or a semicolon may be used to separate a series of clauses. For example, DATA RECORD IS TRANSACTION, RECORD CONTAINS 80 CHARACTERS.

8. A semicolon may be used to separate a series of statements. For example, ADD A TO B; SUBTRACT B FROM C.

9. The word THEN may be used to separate a series of statements. For example, ADD A TO B THEN SUBTRACT B FROM C.

Characters Used for Editing

Editing characters are single characters or specific two-character combinations belonging to the following set:

Character	Meaning
B	space
0	zero
+	plus
-	minus
CR	credit
DB	debit
Z	zero suppression
*	check protection
$	currency sign
,	comma
.	period (decimal point)

(For applications, see the discussion of alphanumeric edited and numeric edited data items in "Data Division.")

Characters Used in Arithmetic Expressions

The characters used in arithmetic expressions are as follows:

Character	Meaning
+	addition
-	subtraction
*	multiplication
/	division
**	exponentiation

Arithmetic expressions are used in the COMPUTE statement and in relation conditions (see "Procedure Division" for more details).

Characters Used for Relation-conditions

A relation character is a character that belongs to the following set:

Character	Meaning
>	greater than
<	less than
=	equal to

Relation characters are used in relation-conditions (discussed in "Procedure Division").

TYPES OF WORDS

A word is composed of a combination of not more than 30 characters chosen from the character set for words. The word cannot begin or end with a hyphen.

Words

The space (blank) is not an allowable character in a word; the space
is a word separator. Wherever a space is used as a word separator, more
than one may be used.

A word is terminated by a space, or by a period, right parenthesis,
comma, or semicolon.

Reserved Words

Reserved words exist for syntactical purposes and must not appear as
user-defined words. However, reserved words may appear as nonnumeric
literals, i.e., a reserved word may be enclosed in quotation marks.
When used in this manner, they do not take on the meaning of reserved
words and violate no syntactical rules.

There are three types of reserved words:

1. Key Words. A key word is a word whose presence is required in a
    COBOL entry. Such words are upper case and underlined in the
    formats given in this publication.

    Key words are of three types:

    a. Verbs such as ADD, READ, and ENTER.

    b. Required words, which appear in statement and entry formats,
        such as the word TO in the ADD statement.

    c. Words that have a specific functional meaning, such as ZERO,
        NEGATIVE, SECTION, TALLY, etc.

2. Optional Words. Within each format, upper case words that are not
    underlined are called optional words because they may appear at the
    user's option. The presence or absence of each optional word in
    the source program does not alter the compiler's translation.
    Misspelling of an optional word, or its replacement by another word
    of any kind, is not allowed.

3. Connectives. There are three types of connectives:

    a. Qualifier connectives, which are used to associate a data-name
        or paragraph-name with its qualifier. The qualifier
        connectives are OF and IN (see "Methods of Data Reference").

    b. Series connectives, which link two or more consecutive
        operands. The series connective is the comma (,).

    c. Logical connectives that are used in compound conditions. The
        logical connectives are AND, OR, AND NOT, and OR NOT (see
        "Conditions").

Note: Abbreviations (such as PIC for PICTURE) are allowed for some
reserved words; the abbreviation is the equivalent of the complete word.
For the formats in which they are allowable, such abbreviations are
shown in the format. The reserved words THRU and THROUGH are
equivalent. In statement formats, wherever the reserved word THRU
appears, the word THROUGH is also allowed.

Names

There are three types of names used in a COBOL program:

1. A data-name is a word that contains at least one alphabetic character and identifies a data item in the Data Division. The following are formed according to the rules for data-names:

   file-names
   index-names
   mnemonic-names
   record-names
   report-names
   sort-file-names
   sort-record-names

2. A condition-name is a name given to a specific value, set of values, or range of values, within the complete set of values that a particular data item may assume. The data item itself is called a conditional variable. The condition-name must contain at least one alphabetic character (see "Data Division" and the discussion of "Special-Names" in "Environment Division").

3. A procedure-name is either a paragraph-name or a section-name. A procedure-name may be composed solely of numeric characters. Two numeric procedure-names are equivalent if, and only if, they are composed of the same number of digits and have the same value (see "Procedure Division"). The following are formed according to the rules for procedure-names:

   library-names
   program-names

Special-Names

Special-names are used in the SPECIAL-NAMES paragraph of the Environment Division. The term special-name refers to a mnemonic-name. A mnemonic-name is a programmer-defined word that is associated in the Environment Division with a function-name: function-names are names with a fixed meaning, defined by IBM.

In the Procedure Division, mnemonic-name can be written in place of its associated function-name in any format where such substitution is valid. The formation of a mnemonic-name follows the rules for formation of a data-name (see "Special-Names" in "Environment Division").

CONSTANTS

A constant is a unit of data whose value is not subject to change. There are two types of constants: literals and figurative constants.

Literals

A literal is a string of characters whose value is determined by the set of characters of which the literal is composed. Every literal belongs to one of two categories, numeric and nonnumeric.

NUMERIC LITERALS: There are two types of numeric literals: fixed-point and floating-point.

A fixed-point numeric literal is defined as a string of characters chosen from the digits 0 through 9, the plus sign, the minus sign, and the decimal point. Every fixed-point numeric literal:

1. must contain from 1 through 18 digits.

2. must not contain more than one sign character. If a sign is used, it must appear as the leftmost character of the literal. If the literal is unsigned, the literal is positive.

3. must not contain more than one decimal point. The decimal point is treated as an assumed decimal point, and may appear anywhere in the literal except as the rightmost character. If the literal contains no decimal point, the literal is an integer.

(See discussion of fixed-point numeric items in "Data Division.")

A floating-point numeric literal is a data item whose potential range of value is too great for fixed-point representation. A floating-point literal must have the form:

[±]mantissa E[±]exponent

A floating-point literal must appear as a continuous string of characters with no intervening spaces. The plus or minus signs preceding the mantissa and exponent are the only optional characters within the format. The mantissa consists of from 1 through 16 digits with a required decimal point.

The exponent is represented immediately to the right of the mantissa by the symbol E, followed by a plus or minus sign (if a sign is given) and one or two digits. The magnitude of the number represented by a floating-point literal must not exceed $.72 \times (10^{76})$. A zero exponent must be written as 0 or 00. It is assumed that an unsigned exponent is positive.

The value of the literal is the product of the mantissa and ten raised to the power given by the exponent. For example, the literal

+.72E+76

has the value

$.72 \times 10^{76}$

(See discussion of floating-point numeric items in "Data Division.")

If the literal conforms to the rules for the formation of numeric literals, but is enclosed in quotation marks, it is a nonnumeric literal.

NONNUMERIC LITERALS: A nonnumeric literal is defined as a string of any allowable characters in the Extended Binary Coded Decimal Interchange Code (EBCDIC) set, excluding the quotation mark character. A nonnumeric literal may be composed of from 1 through 120 characters enclosed in quotation marks. Any spaces within the quotation marks are part of the nonnumeric literal and, therefore, are part of the value. All non-numeric literals are in the alphanumeric category.

Figurative Constants

A <u>figurative constant</u> is a constant to which a specific data-name has been assigned. These data-names are reserved words. Such a data-name must not be enclosed in quotation marks when used as a figurative
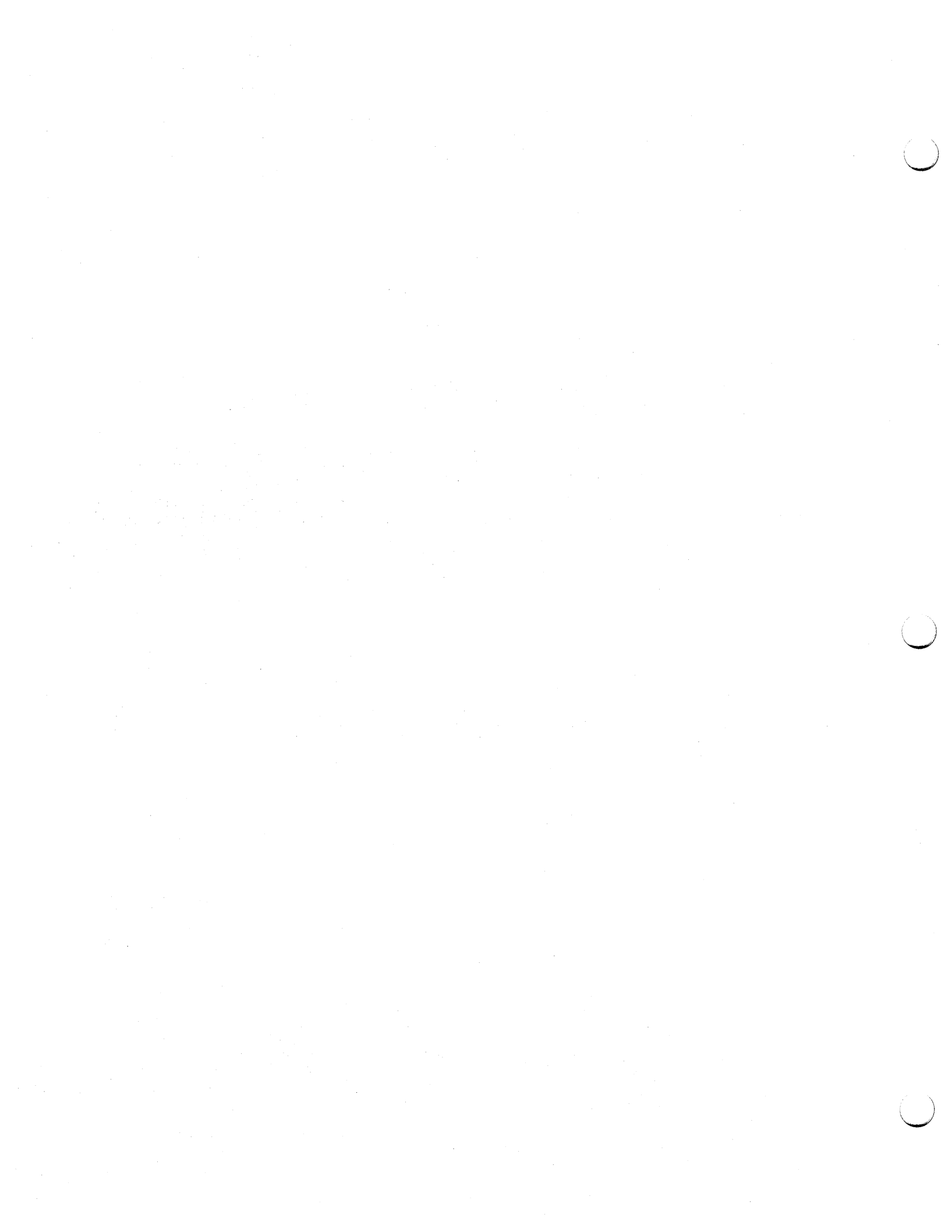
constant. The singular and plural forms of a figurative constant are equivalent and may be used interchangeably.

A figurative constant may be used in place of a literal wherever a literal appears in a format. There is one exception to this rule: if the literal is restricted to numeric characters, only the figurative constant ZERO (ZEROES, ZEROS) is allowed.

The fixed data-names and their meanings are as follows:

ZERO
ZEROES
ZEROS
    Represents the value 0, or one or more occurrences of the character 0, depending on context.

SPACE
SPACES
    Represents one or more blanks or spaces.

HIGH-VALUE
HIGH-VALUES
    Represents one or more occurrences of the character that has the highest value in the computer's collating sequence. The character for HIGH-VALUE is the hexadecimal 'FF'.

LOW-VALUE
LOW-VALUES
    Represents one or more occurrences of the character that has the lowest value in the computer's collating sequence. The character for LOW-VALUE is the hexadecimal '00'.

QUOTE
QUOTES
    Represents one or more occurrences of the quotation mark character. The word QUOTE (QUOTES) cannot be used in place of a quotation mark to enclose a nonnumeric literal.

ALL literal
    Represents one or more occurrences of the string of characters composing the literal. The literal must be either a nonnumeric literal or a figurative constant other than the ALL literal. When a figurative constant is used, the word ALL is redundant and is used for readability only.

SPECIAL REGISTERS

The compiler generates storage areas that are primarily used to store information produced with the use of special COBOL features; these storage areas are called special registers.

TALLY

The word TALLY is the name of a special register whose implicit description is that of an integer of five digits without an operational sign, and whose implicit USAGE is COMPUTATIONAL. The primary use of the TALLY register is to hold information produced by the EXAMINE statement. References to TALLY may appear wherever an elementary data item of integral value may appear (see the "EXAMINE Statement" in "Procedure Division").

LINE-COUNTER

LINE-COUNTER is a numeric counter that is generated by the Report Writer. (For a complete discussion, see "Report Writer.")

PAGE-COUNTER

PAGE-COUNTER is a numeric counter that is generated by the Report Writer. (For a complete discussion, see "Report Writer.")

CURRENT-DATE

CURRENT-DATE is an 8-byte alphanumeric field, valid only as the sending field in a MOVE statement. The format of these eight bytes is MM/DD/YY (month/day/year) or DD/MM/YY (day/month/year).

TIME-OF-DAY

TIME-OF-DAY is a 6-byte external-decimal field, valid only as the sending field in a MOVE statement. The format is HHMMSS (hour, minute, second).

COM-REG

COM-REG is an 11-byte alphanumeric field. This field corresponds to bytes 12 through 22 of the DOS Communication Region. COM-REG is valid only as the sending or receiving field in a MOVE statement. When COM-REG is used as the receiving field in a MOVE statement, the sending field must be 11 bytes in length.

(The use of CURRENT-DATE, TIME-OF-DAY, and COM-REG is explained in the Programmer's Guides (as cited in "Preface").)

SORT-RETURN

SORT-FILE-SIZE

SORT-CORE-SIZE

SORT-MODE-SIZE

The foregoing four registers are used by the Sort feature and are described under "Sort."

NSTD-REELS

NSTD-REELS is a binary field whose PICTURE is S999 and whose maximum value is 255. It is used to indicate the number of reels in an input file with nonstandard labels. This field is initially set to zero. NSTD-REELS is tested each time an OPEN statement is executed for a file with nonstandard labels. If it is nonzero, its contents are used to determine the number of reels to be processed. If it is zero, the number of reels in the file is determined from the ASSIGN clause.

Every COBOL source program is divided into four divisions. Each division must be placed in its proper sequence, and each must begin with a division header.

The four divisions, listed in sequence, and their functions are:

- IDENTIFICATION DIVISION, which names the program.

- ENVIRONMENT DIVISION, which indicates the machine equipment and equipment features to be used in the program.

- DATA DIVISION, which defines the nature and characteristics of data to be processed.

- PROCEDURE DIVISION, which consists of statements directing the processing of data in a specified manner at execution time.

Note: In all formats within this publication, the required clauses and optional clauses (when written) must appear in the sequence given in the format, unless the associated rules explicitly state otherwise.

Structure of the COBOL Program

```
{ IDENTIFICATION DIVISION. }
{ ID DIVISION.            }

PROGRAM-ID.   program-name.

[AUTHOR.   [comment-entry]...]

[INSTALLATION.   [comment-entry]...]

[DATE-WRITTEN.   [comment-entry]...]

[DATE-COMPILED.   [comment-entry]...]

[SECURITY.   [comment-entry]...]

[REMARKS.   [comment-entry]...]

ENVIRONMENT DIVISION.

[CONFIGURATION SECTION.

SOURCE-COMPUTER.   entry

OBJECT-COMPUTER.   entry

[SPECIAL-NAMES.   entry]]

[INPUT-OUTPUT SECTION.

FILE-CONTROL.   {entry}...

[I-O-CONTROL.   entry]]
```

```
DATA DIVISION.

[FILE SECTION.

{file description entry

{record description entry}...}...]

[WORKING-STORAGE SECTION.

[data item description entry]...

[record description entry]...]

[LINKAGE SECTION.

[data item description entry]...

[record description entry]...]

[REPORT SECTION.

{report description entry

{report group description entry}...}...]

PROCEDURE DIVISION [USING identifier-1 [identifier-2]...].

[[DECLARATIVES.

{section-name SECTION.   USE Sentence.

{paragraph-name.   {sentence}...}...}...

END DECLARATIVES.]

{section-name SECTION [priority].]

{paragraph-name.   {sentence}...}...}...
```

Every name used in a COBOL source program must be unique, either because no other name has the identical spelling, or because it is made unique through qualification, subscripting, or indexing.

An identifier is a data-name, unique in itself, or made unique by the syntactically correct combination of qualifiers, subscripts, and/or indexes.

QUALIFICATION

A name may be made unique if the name exists within a hierarchy of names and the name can be singled out by mentioning one or more of the higher levels of the hierarchy. The higher levels are called qualifiers. Qualification is the process by which such a name is made unique.

Qualification is applied by placing after a data-name or a paragraph-name one or more phrases, each composed of a qualifier preceded by IN or OF. IN and OF are logically equivalent. Only one qualifier is allowed for a paragraph-name.

Enough qualification must be mentioned to make the name unique; however, it may not be necessary to mention all levels of the hierarchy. For example, if there is more than one file whose records contain the field EMPLOYEE-NO, yet there is but one file whose records are named MASTER-RECORD, EMPLOYEE-NO OF MASTER-RECORD would sufficiently qualify EMPLOYEE-NO. EMPLOYEE-NO OF MASTER-RECORD OF MASTER-FILE is valid but unnecessary (see discussion of level indicators and level numbers in "Data Division").

The name associated with a level indicator is the highest level qualifier available for a data-name. (A level indicator (FD, SD, RD) specifies the beginning of a file description, sort file description, or report description.) A section-name is the highest (and the only) qualifier available for a procedure-name (see discussion of procedure-names in "Procedure Division"). Thus, level indicator names and section-names must be unique in themselves since they cannot be qualified.

The name of a conditional variable can be used as a qualifier for any of its condition-names. In addition, a conditional variable may be qualified to make it unique.

The rules for qualification follow:

1. Each qualifier must be of a successively higher level, and must be within the same hierarchy as the name it qualifies.

2. The same name must not appear at two levels in a hierarchy so that the name would appear to qualify itself.

3. If a data-name or a condition-name is assigned to more than one data item in a source program, the data-name or condition-name must be qualified each time reference is made to it in the Procedure, Environment, or Data Division (except in the REDEFINES clause where, by definition, qualification is unnecessary). (See the REDEFINES clause in "Data Division.")

4. A paragraph-name must not be duplicated within a section. When a paragraph-name is qualified by a section-name, the word SECTION must not appear. A paragraph-name need not be qualified when referred to within the section in which it appears.

5. A data-name cannot be subscripted when it is being used as a qualifier.

6. A name can be qualified even though it does not need qualification; if there is more than one combination of qualifiers that ensures uniqueness, then any of these combinations can be used.

Although user-defined data-names can be duplicated within the Data Division and Procedure Division, the following rules should be noted:

1. No duplicate section-names are allowed.

2. No data-name can be the same as a section-name or a paragraph-name.

3. Duplication of data-names must not occur in those places where the data-names cannot be made unique by qualification.

SUBSCRIPTING

Subscripts can be used only when reference is made to an individual element within a list or table of' elements that have not been assigned individual data-names (see "Table Handling").

INDEXING

References can be made to individual elements within a table of elements by specifying indexing for that reference. An index is assigned to a given level of a table by using an INDEXED BY clause in the definition of the table. A name given in the INDEXED BY clause is known as an index-name and is used to refer to the assigned index (see "Table Handling").

The reference format provides a standard method for writing COBOL source programs. The format is described in terms of character positions in a line on an input/output medium. Punched cards are the initial input medium to the COBOL compiler. The compiler accepts source programs written in reference format (see Figure 3) and produces an output listing of the source program in the same reference format.

Figure 3. Reference Format

The rules for spacing given in the following discussion of the reference format take precedence over any other specifications for spacing given in this publication.

SEQUENCE NUMBERS

A sequence number, consisting of six digits in the sequence number area, is used to identify numerically each card image to be compiled by the COBOL compiler. The use of sequence numbers is optional.

If sequence numbers are present, they must be in ascending order. An error message is issued when source language input is out of sequence. Sequence checking can be suppressed at compile time by overriding the compiler's default option of checking.

CONTINUATION OF LINES

Any sentence or entry that requires more than one line is continued by starting subsequent line(s) in Area B. These subsequent lines are called continuation lines. The line being continued is called the continued line. If a sentence or entry occupies more than two lines, all lines other than the first and last are both continuation and continued lines.

Reference Format

CONTINUATION OF NONNUMERIC LITERALS

When a nonnumeric literal is continued from one line to another, a
hyphen is placed in column 7 of the continuation line, and a quotation
mark preceding the continuation of the literal may be placed anywhere in
Area B. All spaces at the end of the continued line and any spaces
following the quotation mark of the continuation line and preceding the
final quotation mark are considered part of the literal.

CONTINUATION OF WORDS AND NUMERIC LITERALS

When a word or numeric literal is continued from one line to another,
a hyphen must be placed in column 7 of the continuation line to indicate
that the first nonblank character in Area B of the continuation line is
to follow the last nonblank character on the continued line, without an
intervening space.

AREA A AND AREA B

Area A, columns 8 through 11, is reserved for the beginning of
division headers, section-names, paragraph-names, level indicators, and
certain level numbers. Area B occupies columns 12 through 72.

Division Header

The division header must be the first line in a division. The
division header starts in Area A with the division-name, followed by a
space and the word DIVISION, and a period. If this program is to be
called, a space and a USING clause may follow the words PROCEDURE
DIVISION. No other text may appear on the same line as the division
header.

Section Header

The name of a section starts in Area A of any line following the
division header. The section-name is followed by a space, the word
SECTION, and a period. If program segmentation is desired, a space and
a priority number may follow the word SECTION. No other text may appear
on the same line as the section-header, except USE and COPY sentences.

Note: Although USE and COPY may appear in the Declaratives portion of
the Procedure Division, only USE is restricted to the Declaratives
portion. COPY may be used elsewhere in the COBOL program.

Paragraph-names and Paragraphs

The name of a paragraph starts in Area A of any line following the
division header. It is followed by a period followed by a space.

A paragraph consists of one or more successive sentences. The first
sentence in a paragraph begins anywhere in Area B of either the same
line as paragraph-name or the immediately following line. Each
successive line in the paragraph starts anywhere in Area B.

Level Indicators and Level Numbers

In those Data Division entries that begin with a level indicator, the level indicator begins in Area A followed in Area B by its associated file-name and appropriate descriptive information.

In those data description entries that begin with a level number 01 or 77, the level number begins in Area A followed in Area B by its associated data-name and appropriate descriptive information.

In those data description entries that begin with level numbers 02 through 49, 66, or 88, the level number may begin anywhere in Area A or Area B, followed in Area B by its associated data-name and descriptive information.

Blank Lines

A blank line is one that contains nothing but spaces from column 7 through column 72, inclusive. A blank line may appear anywhere in the source program, except immediately preceding a continuation line.

Comment Lines

Explanatory comments may be inserted on any line within a source program by placing an asterisk in the Continuation Area of the line. Any combination of the characters from the EBCDIC set may be included in Areas A and B of that line. The asterisk and the characters will be produced on the source listing but serve no other purpose. (Also, see the NOTE statement in "Compiler Directing Statements" in "Procedure Division").

FORMAT NOTATION

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

1. All words printed entirely in capital letters are reserved words. These are words that have preassigned meanings in COBOL. In all formats, words in capital letters represent an actual occurrence of those words. If any such word is incorrectly spelled, it will not be recognized as a reserved word and may cause an error in the program.

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

3. The characters +, -, <, >, =, when appearing in formats, although not underlined, are required when such formats are used.

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

5. Words that are printed in lower-case letters represent information to be supplied by the programmer. All such words are defined in the accompanying text.

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

7. Certain entries in the formats consist of a capitalized word(s) followed by the word "Clause" or "Statement." These designate clauses or statements that are described in other formats, in appropriate sections of the text.

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

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

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

11. Comments, restrictions, and clarifications on the use and meaning of every format are contained in the appropriate portions of the text.
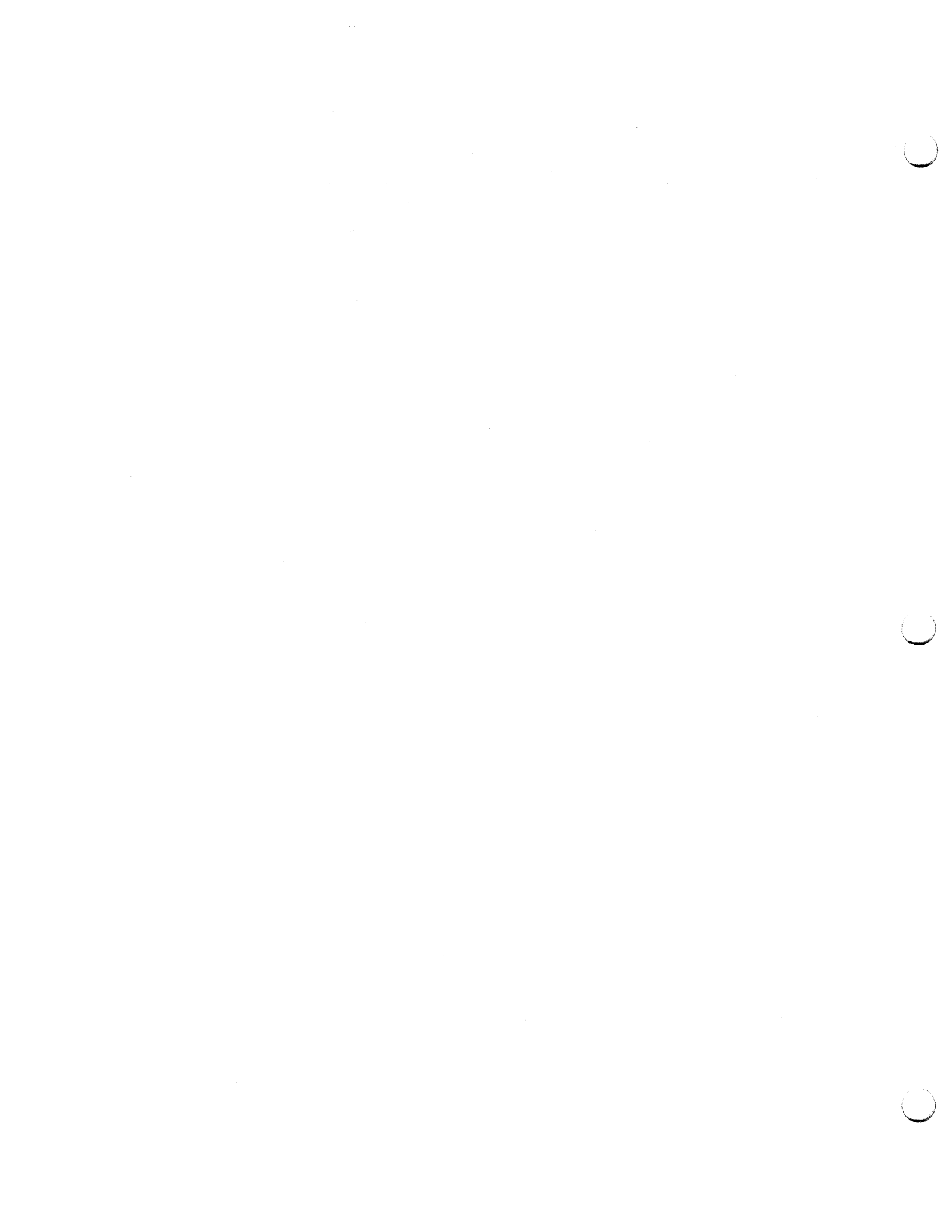
- IDENTIFICATION DIVISION

- ENVIRONMENT DIVISION -- FILE PROCESSING SUMMARY

- ORGANIZATION OF THE ENVIRONMENT DIVISION

- ENVIRONMENT DIVISION -- CONFIGURATION SECTION
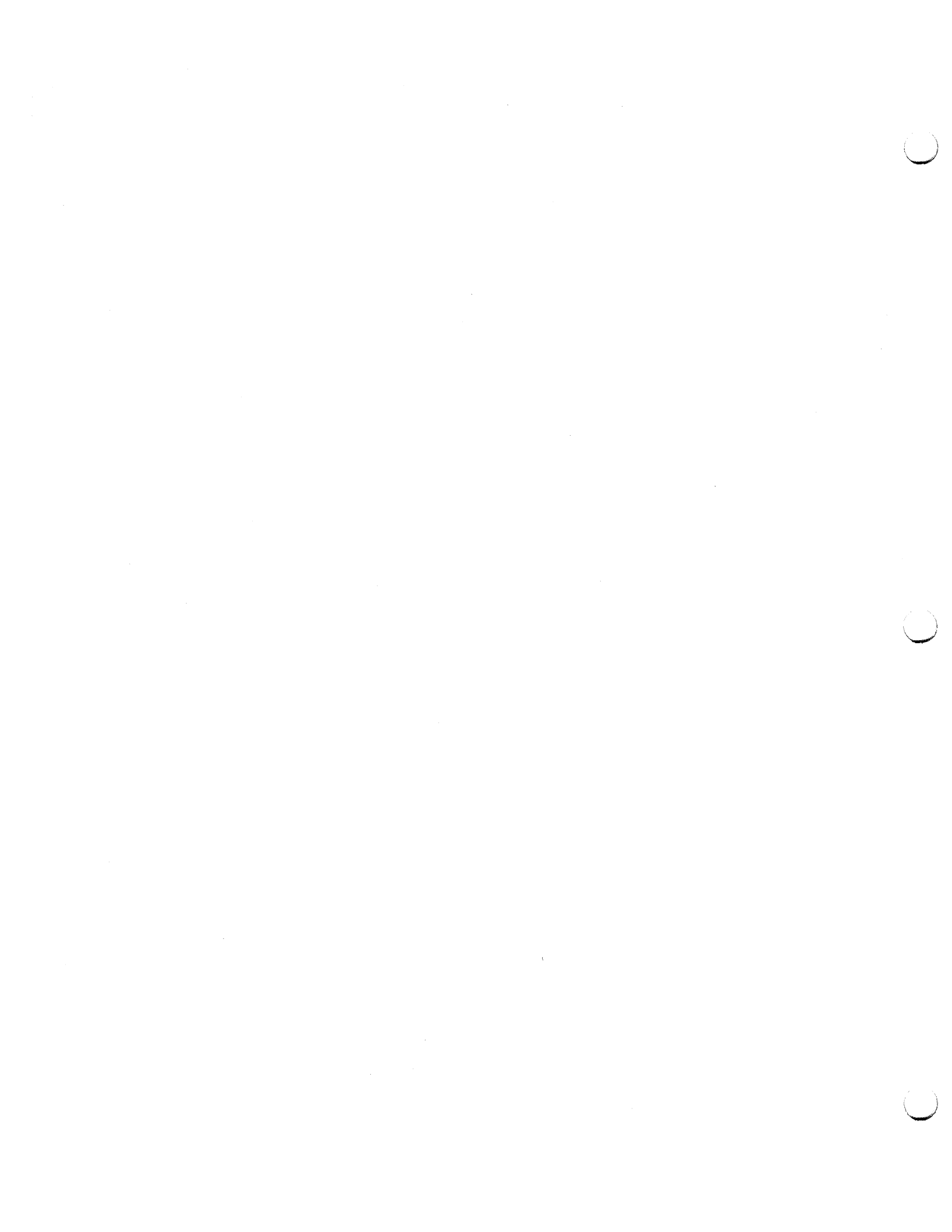
- ENVIRONMENT DIVISION -- INPUT-OUTPUT SECTION

The Identification Division is the first division of a COBOL program. It identifies the source program and the object program. A source program is the initial problem program; an object program is the output from a compilation.

In addition, the user may include the date the program is written, the date the compilation of the source program is accomplished, etc., in the paragraphs shown.

Structure of the Identification Division

    { IDENTIFICATION DIVISION. }
    { ID DIVISION.   }

    PROGRAM-ID.   program-name.

    [AUTHOR.   [comment-entry]...]

    [INSTALLATION.   [comment-entry]...]

    [DATE-WRITTEN.   [comment-entry]...]

    [DATE-COMPILED.   [comment-entry]...]

    [SECURITY.   [comment-entry]...]

    [REMARKS.   [comment-entry]...]

Specific paragraph-names identify the type of information contained in the paragraph. The name of the program must be given in the first paragraph, which is the PROGRAM-ID paragraph. The other paragraphs are optional. If included, they must be presented in the order shown. However, this compiler will accept them in any order.

The Identification Division must begin with the reserved words IDENTIFICATION DIVISION followed by a period. Each comment-entry may be any combination of characters from the EBCDIC set, organized to conform to sentence and paragraph structure. This compiler will accept ID DIVISION followed by a period as a substitute for the standard division header.

PROGRAM-ID Paragraph

The PROGRAM-ID paragraph gives the name by which a program is identified.

```
r--------------------------------------------------------------------------1
|                              Format                                      |
|--------------------------------------------------------------------------|
|                                                                          |
|    PROGRAM-ID.   program-name.                                           |
|                                                                          |
L--------------------------------------------------------------------------J
```

The PROGRAM-ID paragraph contains the name of the program and must be present in every program.

Program-name identifies the object program to the control program. Program-name must conform to the rules for formation of a procedure-name. The first eight characters of program-name are used as the identifying name of the program and should therefore be unique as a program-name.

Since the system expects the first character of program-name to be alphabetic, the first character, if it is numeric, will be converted as follows:

        0 to J

        1-9 to A-I

Since the system does not include the hyphen as an allowable character, the hyphen is converted to zero if it appears as the second through eighth character of the name.

DATE-COMPILED Paragraph

The DATE-COMPILED paragraph provides the compilation date on the source program listing.


```
┌─────────────────────────────────────────────────────────────────────┐
│                             Format                                    │
├─────────────────────────────────────────────────────────────────────┤
│                                                                       │
│    DATE-COMPILED.   [comment-entry]                                   │
│                                                                       │
└─────────────────────────────────────────────────────────────────────┘
```


The paragraph-name DATE-COMPILED causes the current date to be inserted during program compilation. If a comment-entry is present, it is replaced with the current date.

In COBOL, all aspects of the total data processing problem that depend on the physical characteristics of a specific computer are given in one portion of the source program known as the Environment Division. Thus, a change in computers entails major changes in this division only. The primary functions of the Environment Division are to describe the computer system on which the object program is run and to establish the necessary links between the other divisions of the source program and the characteristics of the computer.

The exact contents of the Environment Division depend on the method used to process files in the COBOL program. Before the language elements used in the Environment Division can be discussed meaningfully, some background in the file processing techniques available to the COBOL user must be given.

Each combination of data organization and access method specified in the COBOL language is defined as a file-processing technique. The file-processing technique to be used for a particular file is determined by the data organization of that file and whether the access method is sequential or random. Table 3 summarizes the file-processing techniques.

DATA ORGANIZATION

Three types of data organization are made available to Disk Operating System COBOL users: sequential, direct, and indexed. The means of creating or retrieving logical records in a file differ, depending on which type of data organization exists (organization being the structure of data on a physical file). Each type of data organization is incompatible with the others. Organization of an input file must be the same as the organization of the file when it was created.

Sequential Data Organization

When sequential data organization is used, the logical records in a file are positioned sequentially in the order in which they are created and are read sequentially in the order in which they were created (or in sequentially reversed order if the REVERSED option of the OPEN statement is written for tape files). Such a file organization is referred to in this publication as standard sequential organization.

This type of data organization must be used for tape or unit-record files and may be used for files assigned to mass storage devices. No key is associated with records on a sequentially organized file.

Direct Data Organization

When direct data organization is used, the positioning of the logical records in a file is controlled by the user through the specification of an ACTUAL KEY defined in the Environment Division. The ACTUAL KEY has two components. The first is a track identifier which identifies the relative or actual track at which a record is to be placed or at which the search for a record is to begin. The second component is a record

identifier, which serves as a unique logical identifier for a specific
record on the track. Files with direct data organization must be
assigned to mass storage devices.

Indexed Data Organization

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

ACCESS METHODS

Two access methods are available to users of DOS COBOL: sequential
access and random access.

Sequential access is the method of reading and writing records of a
file in a serial manner; the order of reference is implicitly determined
by the position of a record in the file.

Random access is the method of reading and writing records in a
programmer-specified manner; the control of successive references to the
file is expressed by specifically defined keys supplied by the user.

ACCESSING A SEQUENTIAL FILE

A standard sequential file may be accessed only sequentially, i.e.,
records are read or written in order.

ACCESSING A DIRECT FILE

Direct files may be accessed both sequentially and randomly. Records
can be retrieved sequentially; they can be created, retrieved, updated,
or added randomly.

Sequential Access

When reading a direct file sequentially, records are retrieved in
logical sequence; this logical sequence corresponds exactly to the
physical sequence of the records.

If the ACTUAL KEY clause is specified, the key associated with the
record will be placed in the field specified by the ACTUAL KEY clause,
for each READ statement executed.

Random Access

When accessing a direct file randomly, the ACTUAL KEY clause is
required.

The system uses the ACTUAL KEY to determine which track a particular record is on and to locate the record on that track. If the record is found, the data portion of the record is read, or, for a rewrite operation, replaced by a new record. If during a READ operation, the desired record cannot be found on the specified track, an invalid key condition is said to exist.

Unless APPLY EXTENDED-SEARCH is used, only the specified track is searched for the desired record. When APPLY EXTENDED-SEARCH is used, the system searches the entire cylinder for the record if the desired record cannot be found.

For a write operation, the system, after locating the track, searches for the last record on the track, and writes the new record (with control fields, including a key field equal to the identifier found within the ACTUAL KEY field) after the last record.

When a direct file is being created, OPEN initializes the capacity records (R0) on all the tracks of the file. Therefore, a WRITE statement issued for an output file is processed in the same manner as a WRITE statement that adds a record to an input-output file.

ACCESSING AN INDEXED FILE

An indexed file may be accessed both sequentially and randomly. Records can be created, retrieved, or updated sequentially; they can be retrieved, updated, or added randomly.

The same indexed file may not be created and read or updated within the same program.

Sequential Access

An indexed file may only be created sequentially. When creating an indexed file, the RECORD KEY clause must be specified. It is used to indicate the location of the key within the record itself. The NOMINAL KEY clause may be specified. Records are arranged in the file in the order in which they are written.

To retrieve or update an indexed file sequentially, the RECORD KEY clause must be specified. Records are read in the order in which they are arranged in the file. Logically, this corresponds to the sequence of keys, which must be in collating sequence at the time the file is created. If record retrieval is to begin at other than the first record, the NOMINAL KEY clause must be specified and a START statement must be executed before the first READ operation for that file.

Random Access

To retrieve or update an indexed file randomly, both the NOMINAL KEY and RECORD KEY clauses are required. A record is considered "found" when the value associated with the data-name specified in the NOMINAL KEY clause is equal to the value of the RECORD KEY for the record. When adding a record to or updating a randomly accessed indexed file, the value associated with the RECORD KEY must be identical to that of the NOMINAL KEY field.

Appendix B contains three sample COBOL programs that illustrate:

1. Creation of a direct file

   (Figure 2 contains a sample COBOL program illustrating random retrieval and updating of a direct file.)

2. Creation of an indexed file

3. Random retrieval and updating of an indexed file

Table 3. Summary of File-Processing Techniques

DOS Organization	Device Type	Access	Organization
DTFCD	Reader	[SEQUENTIAL]	standard sequential
DTFCD	Punch	[SEQUENTIAL]	standard sequential
DTFPR	Printer	[SEQUENTIAL]	standard sequential
DTFMT	Tape	[SEQUENTIAL]	standard sequential
DTFSD	Mass Storage	[SEQUENTIAL]	standard sequential
DTFDA	Mass Storage	[SEQUENTIAL]	direct
DTFDA	Mass Storage	RANDOM	direct
DTFIS	Mass Storage	[SEQUENTIAL]	indexed
DTFIS	Mass Storage	RANDOM	indexed

ORGANIZATION OF THE ENVIRONMENT DIVISION

The Environment Division must begin in Area A with the heading ENVIRONMENT DIVISION followed by a period.

The Environment Division is divided into two sections: the Configuration Section and the Input-Output Section.   When written, the sections and paragraphs must be in the sequence shown.

Structure of the Environment Division

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER paragraph

OBJECT-COMPUTER paragraph

[SPECIAL-NAMES paragraph]

[INPUT-OUTPUT SECTION.

FILE-CONTROL paragraph

[I-O-CONTROL paragraph]]

ENVIRONMENT DIVISION -- CONFIGURATION SECTION

The Configuration Section deals with the overall specifications of computers. It is divided into three paragraphs: the SOURCE-COMPUTER paragraph, which describes the computer on which the source program is compiled; the OBJECT-COMPUTER paragraph, which describes the computer on which the object program (the program produced by the COBOL compiler) is executed; and, optionally, the SPECIAL-NAMES paragraph which relates the function-names used by the compiler to user-specified mnemonic-names.

```
---------------------------------------------------------------------
|                          General Format                            |
|-------------------------------------------------------------------|
|                                                                    |
|   CONFIGURATION SECTION.                                           |
|   SOURCE-COMPUTER. source-computer-entry                           |
|   OBJECT-COMPUTER.   object-computer-entry                         |
|   [SPECIAL-NAMES. special-names-entry]                             |
|                                                                    |
---------------------------------------------------------------------
```

Section-names and paragraph-names must begin in Area A.

The Configuration Section and its associated paragraphs are optional within a COBOL source program.

SOURCE-COMPUTER Paragraph

The SOURCE-COMPUTER paragraph serves only as documentation, and describes the computer upon which the program is to be compiled.

```
---------------------------------------------------------------------
|                            Format                                  |
|-------------------------------------------------------------------|
|                                                                    |
|   SOURCE-COMPUTER.   computer-name.                                |
|                                                                    |
---------------------------------------------------------------------
```

Computer-name may be specified as IBM-360[-model-number] or as IBM-370[-model-number].

The SOURCE-COMPUTER paragraph is treated as comments by the COBOL compiler.

OBJECT-COMPUTER Paragraph

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

```
┌───────────────────────────────────────────────────────────────────┐
│                              Format                               │
├───────────────────────────────────────────────────────────────────┤
│                                                                   │
│   OBJECT-COMPUTER.   computer-name                                │
│                                                                   │
│                                      ⎧ WORDS      ⎫                │
│        [MEMORY SIZE integer          ⎨ CHARACTERS ⎬  ]             │
│                                      ⎩ MODULES    ⎭                │
│                                                                   │
│        [SEGMENT-LIMIT IS priority-number].                        │
│                                                                   │
└───────────────────────────────────────────────────────────────────┘
```

Computer-name is a word of the form IBM-360[-model-number]. Computer-name must be the first entry in the OBJECT-COMPUTER paragraph.

If the configuration implied by computer-name comprises more or less equipment than is actually needed by the object program, the MEMORY SIZE clause permits the specification of the actual subset (or superset) of the configuration.

| The MEMORY SIZE clause is treated as comments by the COBOL compiler.

The SEGMENT-LIMIT clause is discussed in "Segmentation."

Program Product Information -- Version 3

Computer-name may also be specified as IBM-370[-model-number]. If IBM-370 is specified, System/370 instructions are generated by the compiler.

SPECIAL-NAMES Paragraph

The SPECIAL-NAMES paragraph provides a means of relating function-names to user-specified mnemonic-names. The SPECIAL-NAMES paragraph can also be used to exchange the functions of the comma and the period in the PICTURE character string and in numeric literals. In addition, the user may specify a substitution character for the currency symbol ($) in the PICTURE character string.

```
+-------------------------------------------------------------------------+
|                             General Format                              |
+-------------------------------------------------------------------------+
|                                                                         |
|    SPECIAL-NAMES.                                                        |
|                                                                         |
|          [function-name-1 IS mnemonic-name]...                          |
|                                                                         |
|          [function-name-2 [IS mnemonic-name]                            |
|                                                                         |
|              ( ON STATUS IS condition-name-1                            |
|              {                                                          |
|              ( OFF STATUS IS condition-name-2                           |
|                                                                         |
|                          [OFF STATUS IS condition-name-2] )             |
|                                                                 } ]...   |
|                          [ON STATUS IS condition-name-1]  )             |
|                                                                         |
|          [CURRENCY SIGN IS literal] [DECIMAL-POINT IS COMMA].           |
|                                                                         |
+-------------------------------------------------------------------------+
```

   Function-name-1 may be chosen from the following list:

      SYSLST
      SYSPCH
      SYSPUNCH
      SYSIPT
      CONSOLE
      C01 through C12
      CSP
      S01
      S02
      literal

   If SYSLST, SYSPCH, SYSPUNCH, SYSIPT, or CONSOLE are specified, the
associated mnemonic-names may be used in ACCEPT and DISPLAY statements.
Each of these function-names may appear only once in the SPECIAL-NAMES
paragraph.

   If C01 through C12, CSP, S01, or S02 are specified, the associated
mnemonic-names may be used in a WRITE statement with the BEFORE/AFTER
ADVANCING option. These function-names are the carriage control
characters shown in Table 4.

Table 4. Choices of Function-name-1 and Action Taken

Function-name-1	Action Taken
CSP	suppress spacing
C01 through C09	skip to channel 1 through 9, respectively
C10 through C12	skip to channel 10, 11, 12, respectively
S01, S02	pocket select 1 or 2 on the IBM 1442, and P1 or P2 on the IBM 2540.

The use of a <u>literal</u> indicates that <u>function-name-1</u> identifies Report Writer output. The <u>mnemonic-name</u> should appear in a CODE clause in a report description entry (RD) (see "Report Writer"). One such SPECIAL-NAMES entry may be given for each report defined in a program. The specified literal must be a one-character nonnumeric literal.

<u>Function-name-2</u> is used to define a one-byte switch and may be specified as UPSI-0 through UPSI-7. These switches represent the User Program Status Indicator bits in the DOS communications region (see <u>IBM System/360 Disk Operating System: System Control and System Service Programs</u>, Form C24-5036). The status of the switch is specified by a <u>condition-name</u> and interrogated by testing it. One condition-name may be associated with the ON status; another may be associated with the OFF status (see "Switch-Status Condition"). One condition-name must be associated with function-name-2. A mnemonic-name, a second condition-name, or both may be associated with the function-name-2 as well. The condition-names represent the equivalent of level-88 items where UPSI-n or mnemonic-name may be considered the conditional variable.

Either UPSI-n or mnemonic-name may be used for qualification of the condition-name.

Note: This compiler does not support hardware switches. However, the switch status function is provided as an extension to support the system UPSI switches, which are program switches that serve the function of hardware switches.

The <u>literal</u> which appears in the CURRENCY SIGN clause is used in the PICTURE clause to represent the currency symbol. The literal must be nonnumeric and is limited to a single character which must not be any of the following characters:

1.   digits 0 through 9

2.   alphabetic characters A, B, C, D, P, R, S, V, X, Z, or the space

3.   special characters * + - , . ; ( ) " or '

If the CURRENCY SIGN clause is not present, only the $ can be used as the currency symbol in the PICTURE clause.

Note: If the CURRENCY SIGN clause specifies E, then floating point representation cannot be specified.

The clause DECIMAL-POINT IS COMMA means that the function of the comma and the period are exchanged in PICTURE character strings and in numeric literals.

ENVIRONMENT DIVISION -- INPUT-OUTPUT SECTION

The Input-Output Section deals with the definition of each file, the identification of its external storage media, the assignment of the file to one or more input/output devices and with information needed for the most efficient transmission of data between the media and the object program. The section is divided into two paragraphs: the FILE-CONTROL paragraph, which names and associates the files used in the program with the external media; and the I-O-CONTROL paragraph, which defines special input/output techniques.

```
r----------------------------------------------------------------------1
|                           General Format                             |
|----------------------------------------------------------------------|
|                                                                      |
|   [INPUT-OUTPUT SECTION.                                             |
|   FILE-CONTROL. {file-control-entry} ...                            |
|   [I-O-CONTROL. input-output-control-entry]]                        |
|                                                                      |
L----------------------------------------------------------------------J
```

FILE-CONTROL PARAGRAPH

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

```
r----------------------------------------------------------------------1
|                           General Format                             |
|----------------------------------------------------------------------|
|                                                                      |
|   FILE-CONTROL.                                                      |
|                                                                      |
|        {SELECT Clause                                               |
|        ASSIGN Clause                                                |
|        [RESERVE Clause]                                             |
|        [FILE-LIMIT Clause]                                          |
|        [ACCESS MODE Clause]                                         |
|        [PROCESSING MODE Clause]                                     |
|        [ACTUAL KEY Clause]                                          |
|        [NOMINAL KEY Clause]                                         |
|        [RECORD KEY Clause]                                          |
|        [TRACK-AREA Clause].}...                                     |
|                                                                      |
L----------------------------------------------------------------------J
```

SELECT Clause

The SELECT clause is used to name each file in a program.

```
r---------------------------------------------------------------------1
|                            Format                                   |
|---------------------------------------------------------------------|
|                                                                     |
|   SELECT [OPTIONAL] file-name                                       |
|                                                                     |
L_____J
```

Each file used in the program must be named once and only once as a
file-name following the key word SELECT.

Each file named in a SELECT clause must have a File Description (FD)
entry or Sort File Description (SD) entry in the Data Division.

The key word OPTIONAL may be specified only for input files accessed
sequentially. It is required for input files that are not necessarily
present each time the object program is executed. When a file is not
present at object time, the first READ statement for that file causes
the imperative-statement following the key words AT END to be executed.
However, OPTIONAL need not be specified and will be treated as a
comment, since this function is performed through the ASSGN control
statement with the IGN parameter.

ASSIGN Clause

The ASSIGN clause is used to assign a file to an external medium.

```
r---------------------------------------------------------------------1
|                            Format                                   |
|---------------------------------------------------------------------|
|                                                                     |
|   ASSIGN TO [integer] system-name-1 [system-name-2] ...            |
|                                                                     |
|                          (REEL)                                    |
|        [FOR MULTIPLE      {    }    ]                               |
|                          (UNIT)                                    |
|                                                                     |
L_____J
```

Integer indicates the number of input/output units for a given medium
assigned to file-name. Since the number of units is determined at
program execution time (see IBM System/360 Disk Operating System:
System Control and System Service Programs, Form C24-5036), the standard
definition given above is not the action taken by this compiler.

When specified for files with standard labels or for unlabeled output
tape files, the integer option is treated as comments. When integer is
specified as greater than one for unlabeled input tape files, then at
the end of every reel a message is issued to the operator asking whether
or not end-of-file has been reached. It is the user's responsibility to
provide the operator with correct information as to the number of reels
in the file.

For multivolume input files with nonstandard labels, the integer
option is required. For such files, the compiler is unable to
distinguish between end-of-volume and end-of-file and, therefore, cannot
determine the number of reels in the file. Therefore, for input files
with nonstandard labels, the integer option is used to determine the
number of reels in the file. This number can be overridden by storing a
nonzero integer in special register NSTD-REELS before opening the file.
If NSTD-REELS is set to zero at the time the file is opened, integer is
used. If integer is not specified, the system assumes that the file is
contained on one reel.

All files used in a program must be assigned to an external medium.
System-name specifies a device class, a particular device, the
organization of data upon this device, and the external name of the
file. Any system-names beyond the first are treated as comments.

FOR MULTIPLE REEL/UNIT is applicable whenever the number of tape
units or mass storage devices assigned might be less than the number of
reels or units in the file. However, this clause need not be specified.
The system will automatically handle volume switching for sequentially
accessed files. All volumes must be mounted for randomly accessed
files. Therefore, when this clause is specified, it is treated as
comments.

System-name has the following structure:

SYSnnn-class-device-organization[-name]

where:

nnn is a three-digit number between 000 and 221. This field represents
the symbolic unit to which the file is assigned.

class is a two-digit field that represents the device class. The
allowable combinations of characters are:

DA for mass storage
UT for utility
UR for unit record

Files assigned to DA devices may have standard sequential or direct
organization. When organization is direct, access may be either
sequential or random.

Files assigned to DA devices may also have indexed organization.
When organization is indexed, access may be either sequential or random.

Files assigned to UT or UR devices must have standard sequential
organization.

device is a four- or five-digit field that represents a device number.
Device number is used to specify a particular device within a device
class.

The allowable devices for any given device class are as follows:

Mass storage (DA) 2311, 2314, 2321

Utility (UT) 2400, 2311, 2314, 2321

Unit record (UR) 1442R, 1442P, 1403, 1404 (continuous forms only), 1443,
2501, 2520R, 2520P, 2540R, 2540P

(R indicates reader, P indicates punch)

Note: Sort input, output, and work files may be assigned to any utility
device except a 2321 (see "Sort").

organization is a one-character field that specifies file organization.
The letters that may be specified for each type of file are as follows:

S   for standard sequential files
A   for direct files -- actual track addressing
D   for direct files -- relative track addressing
U   for direct files using REWRITE statement* -- actual track addressing
W   for direct files using REWRITE STATEMENT* -- RELATIVE TRACK
      ADDRESSING
I   for indexed files

_____

* When the file is opened as INPUT or OUTPUT, however, U is the
  equivalent of A, W is the equivalent of D.

Table 5 can be used to determine the correct choice of the organization
field in system-names.

name is a one- to seven-character field specifying the external-name by
which the file is known to the system. If specified, it is the name
that appears in the file-name field of the VOL, DLBL, or TLBL job
control statement (see the appropriate Programmer's Guide). If name is
not specified, the symbolic unit (SYSnnn) is used as the external-name.
The field must be specified if more than one file is assigned to the
same symbolic unit.

Program Product Information -- Version 3

      Note: ASCII considerations for the ASSIGN clause are given in
      Appendix E.

      For Version 3, the following additional system devices are
      allowable:

      Mass storage (DA)   2319, 3330
      Utility (UT)   2319, 3330, 3410, 3420
      Unit Record (UR)   3211, 3505, 3525P, 3525R, 3525W, 3525M

      For the Version 3 DA and UT devices (2319, 3330, 3410, 3420), as
      well as for the UR 150-character printer (3211), these numbers can
      be specified in the device field of system-name. For these devices,
      the valid entries for the other fields in system-name are unchanged.

      For the 3505 card reader, system-name has the following format:

                          ⎛ S[R] ⎞
          SYSnnn-UR-3505- ⎨   ⎬ [-name]
                          ⎝ O   ⎠

      The SYSnnn and name fields have the same valid entries as other
      devices.

      For the organization field, the following considerations apply:

      S[R] specifies standard sequential card reading. The optional R
           field specifies RCE (Read Column Eliminate) card reading. When

R is specified, the user can indicate by program control that some card columns are to be ignored when reading data for a particular job. (See the section "RCE and OMR Format Descriptor" for a more complete discussion.) When the R field is omitted, RCE card reading may not be specified.

O   specifies Optical Mark Reading (OMR). When O is specified, then if at object time the device reads a card with a marginal mark, a wear mark, or a poor erasure, the substitution character (hexadecimal "3F") is placed in the defective column and in column 80 (an 80-character buffer is always provided). Thus, if the programmer specifies an 80-character COBOL record, defines column 80 as PIC 9 COMP-3, and then compares column 80 with the numeric literal 3, he is able to identify the defective cards for special handling. Internal-decimal 3 has the bit configuration of hexadecimal "3F". (Also, see the section "RCE and OMR Format Descriptor" for a further discussion.)

For the 3525 card punch with special features, system-name has the following format:

$$
SYSnnn-UR-3525 \begin{Bmatrix} P \\ R \\ W \\ M \end{Bmatrix} - \begin{Bmatrix} S[R] \\ V[R] \\ X[R] \\ Y[R] \\ T \\ Z \end{Bmatrix} [-name]
$$

Note: The optional R code in the organization field is valid only when the device is specified as 3525R.

The name field has the same valid entries as for other devices.

The SYSnnn field, for 3525 files that do not utilize combined function processing, has the same valid entries as other devices.

The SYSnnn field has special considerations when combined function card processing is used. For each associated logical file within the combined function structure there must be a separate SELECT sentence; each such associated logical file must be specified with the same SYSnnn field. (See Appendix G: 3525 Combined Function Processing for a more detailed discussion.)

For the device field, the following entries are valid:

        3525R   for a card read file
        3525P   for a card punch file
        3525W   for a 2-line card print file
        3525M   for a multiline card print file

For the organization field, depending on the device field, the following entries are valid:

        3525R   S[R] for sequential card read files
        (reader)   V[R] for read/print associated files
                    X[R] for read/punch/print associated files
                    Y[R] for read/punch associated files

                    Note: the optional R field specifies RCE (Read Column Eliminate) card reading. (See "RCE and OMR Format Descriptor" for further discussion.)

3525P (punch)	S	for sequential card punch files
	T	for punch-and-interpret files (see Note)
	X	for read/punch/print associated files
	Y	for read/punch associated files
	Z	for punch/print associated files

Note: The T field denotes a normal punched output file for which the graphically printable punched characters are also printed on print lines 1 and 3 of the card. Line 1 contains the first 64 characters, left justified; line 3 contains the last 16 characters, right justified.

3525W (2-line print)	S	for sequential 2-line print files
	V	for read/print associated files
	X	for read/punch/print associated files
	Z	for punch/print associated files

3525M (multi-line print)	S	sequential multiline print files
	V	for read/print associated files
	X	for read-punch-print associated files
	Z	for punch/print associated files

RCE AND OMR FORMAT DESCRIPTOR

When the user specifies O (for Optical Mark Read) or R (for Read Column Eliminate) in the organization field of system-name, then at object time he must provide a format descriptor as the first card(s) in his data deck. If the format descriptor is missing for such files, a message is issued to the operator, and the job is terminated.

The format descriptor must be the first card(s) in the data deck. Column 1 of the first card must be blank. The keyword FORMAT must be punched in columns 2 through 7. Column 8 must be blank. Columns 9 through 71 can contain the parameters that specify which columns of the data cards are to be read in OMR or RCE mode. Continuation cards are valid. A continuation code must be placed in column 72 of the preceding card. Parameters may then be continued, beginning in column 16 of the continuation card. Comments, if used, must follow the last operand on each card by at least one blank space, and continuation card restrictions must be observed.

The format of the format descriptor is as follows:

Col.

```
12....7.9............
 ||    | |
 ||    | |
 ||    | |
 VV    V V
 FORMAT (N1,N2)[,(N3,N4)]...
```

N1, N2, N3, and N4 may be any decimal integers from 1 through 80.
However, N2 must be greater than or equal to N1. N3 must be greater
than or equal to N3. In addition, for OMR processing, N1 and N2
must be both even or both odd, N3 and N4 must be both even or both
odd, and N3 - N2 must be greater than or equal to 2.

In OMR mode, the user establishes which columns are to be read in
OMR mode. For example, if the user wishes to read columns 1, 3, 5,
7, 9 and 70, 72, 74, 76, 78, 80 in OMR mode, the following format
descriptor is valid:

FORMAT (1,9),(70,80)

In RCE mode, the user specifies those columns which are not to be
read. For example, if the user chooses to eliminate columns 20
through 30, and columns 52 through 73, the following format
descriptor is valid:

FORMAT (20,30),(52,73)

Table 5. Values of Organization Field for File Organization

Device Type	ACCESS	File Organization	Track Addressing	Organization Field in System-name
tape, punch, reader, printer	[SEQUENTIAL]	standard sequential	--	S
mass storage device	[SEQUENTIAL]	standard sequential	--	S
mass storage device	[SEQUENTIAL]	direct	actual	A
			relative	D
mass storage device	RANDOM	direct	actual	A
			relative	D
mass storage device	RANDOM	direct (REWRITE)	actual	U
			relative	W
mass storage device	[SEQUENTIAL]	indexed	--	I
mass storage device	RANDOM	indexed	--	I

Program Product Information -- Version 3

● Table 5.1. Values of Organization Field for File Organization

For Version 3 devices, the following entries are valid			
Device Type	ACCESS	File Organization	Organization Field
UR and UT (except 3505, 3525)	[SEQUENTIAL]	standard sequential	S
UR 3505, 3525R (without OMR or RCE)	[SEQUENTIAL]	standard sequential	S
UR 3505 (with OMR)	[SEQUENTIAL]	standard sequential	O
UR 3505, 3525R (with RCE)	[SEQUENTIAL]	standard sequential	SR
UR 3525R, 3525P, 3525W, 3525M	[SEQUENTIAL]	standard sequential	S
UR 3525P punch-interpret file	[SEQUENTIAL]	standard sequential	T
UR 3525R, 3525W, 3525M read/print associated file	[SEQUENTIAL]	standard sequential	V
UR 3525R (with RCE) read/print associated file	[SEQUENTIAL]	standard sequential	VR
UR 3525R, 3525P, 3525W, 3525M read/punch/print associated file	[SEQUENTIAL]	standard sequential	X
UR 3525R (with RCE) read/punch/print associated file	[SEQUENTIAL]	standard sequential	XR
UR 3525R, 3525P read/punch associated file	[SEQUENTIAL]	standard sequential	Y
UR 3525R (with RCE) read/punch associated file	[SEQUENTIAL]	standard sequential	YR
UR 3525P, 3525W 3525M punch/print associated file	[SEQUENTIAL]	standard sequential	Z
DA (mass storage) devices	Entries valid for Version 2 are valid for Version 3		

RESERVE Clause

The RESERVE clause allows the user to modify the number of
input/output areas (buffers) allocated by the compiler for a standard
sequential file or (Version 3 only) a sequentially accessed indexed
file.

```
+-----------------------------------------------------------------------+
|                               Format                                  |
+-----------------------------------------------------------------------+
|                                                                       |
|                 ( integer )               [ AREA  ]                   |
|   RESERVE       {         }    ALTERNATE  [       ]                   |
|                 ( NO      )               [ AREAS ]                   |
|                                                                       |
+-----------------------------------------------------------------------+
```
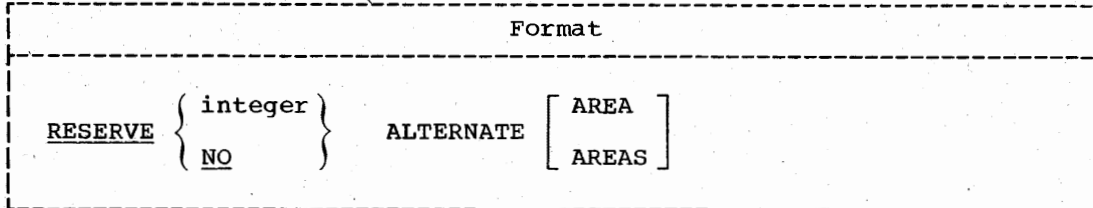
This clause may be specified only for a standard sequential file.

Integer must have a value of 1.

A minimum of one buffer is required for a file. If this clause is
omitted or if 1 is specified, one additional buffer is assumed.

If NO is specified, no additional buffer areas are reserved aside
from the minimum of one.

Program Product Information -- Version 3

For Version 3, the RESERVE clause may specify one additional buffer
for a sequentially accessed indexed file.

Combined function file processing considerations for the RESERVE
clause are given in Appendix G.

FILE-LIMIT Clause

The FILE-LIMIT clause serves only as documentation, and is used to specify the beginning and the end of a logical file on a mass storage device.

```
+------------------------------------------------------------------+
|                             Format                               |
+------------------------------------------------------------------+
|                                                                  |
| ( FILE-LIMIT IS  )  ( data-name-1 )        ( data-name-2 )       |
| (                )  (             )  THRU   (            )        |
| ( FILE-LIMITS ARE )  ( literal-1   )        ( literal-2   )      |
|                                                                  |
|        ( data-name-3 )        ( data-name-4 )                    |
|      [ (            )  THRU   (            ) ] ...               |
|        ( literal-3   )        ( literal-4   )                    |
|                                                                  |
+------------------------------------------------------------------+
```

The logical beginning of a mass storage file is the address specified as the first operand of the FILE-LIMIT clause; the logical end of a mass storage file is the address specified as the last operand of the FILE-LIMIT clause. Because file boundaries are determined at execution time from the control cards, this clause need not be specified and will be treated as comments.

ACCESS MODE Clause

The ACCESS MODE clause defines the manner in which records of a file are to be accessed.

```
+------------------------------------------------------------------+
|                             Format                               |
+------------------------------------------------------------------+
|                                                                  |
|                        ( SEQUENTIAL )                            |
|    ACCESS MODE IS       (            )                           |
|                        ( RANDOM     )                            |
|                                                                  |
+------------------------------------------------------------------+
```

If this clause is not specified, ACCESS IS SEQUENTIAL is assumed. For ACCESS IS SEQUENTIAL, records are placed or obtained sequentially. That is, the next logical record is made available from the file when the READ statement is executed, or the next logical record is placed into the file when a WRITE statement is executed. ACCESS IS SEQUENTIAL may be applied to files assigned to tape, unit-record, or mass storage devices.

For ACCESS IS RANDOM, storage and retrieval are based on an ACTUAL KEY or NOMINAL KEY associated with each record. When the RANDOM option is specified, the file must be assigned to a mass storage device. ACCESS IS RANDOM may be specified when file organization is direct or indexed.

PROCESSING MODE/ACTUAL KEY Clauses

The keyword IS must be specified. However, this compiler allows the keyword IS to be omitted.

PROCESSING MODE Clause

The PROCESSING MODE clause serves only as documentation, and indicates the order in which records are processed.

```
┌─────────────────────────────────────────────────────────────────────┐
│                              Format                                  │
├─────────────────────────────────────────────────────────────────────┤
│                                                                     │
│   PROCESSING MODE IS SEQUENTIAL                                      │
│                                                                     │
└─────────────────────────────────────────────────────────────────────┘
```

This clause is treated as comments, and may be omitted.

ACTUAL KEY Clause

An ACTUAL KEY is a key that is directly usable by the system to locate a logical record on a mass storage device. The ACTUAL KEY is made up of two components.

1. The track identifier, which expresses a track address at which the search for a record, or for a space in which to place a new record, is to begin.

2. The record identifier, which serves as a unique identifier for the record and is associated with the record itself.

When processing a randomly accessed direct file, the programmer is responsible for providing the ACTUAL KEY for each record to be processed.

```
┌─────────────────────────────────────────────────────────────────────┐
│                              Format                                  │
├─────────────────────────────────────────────────────────────────────┤
│                                                                     │
│   ACTUAL KEY IS data-name                                           │
│                                                                     │
└─────────────────────────────────────────────────────────────────────┘
```

Records are accessed randomly and are processed in the order in which they are accessed.

The ACTUAL KEY clause must be specified for direct files when ACCESS IS RANDOM is specified.

The ACTUAL KEY clause may be specified when reading direct files sequentially.

When a SEEK statement is executed, the contents of data-name are used to locate a specific mass storage record area.

When a READ statement is executed, a specific logical record (located by the system using the contents of data-name) is made available from the file.

When a WRITE statement is executed, the given logical record is written at a specific location in the file.

At file creation time, when no more room remains on a given track, a standard error occurs, and the user must provide a USE AFTER STANDARD ERROR declarative routine to update the track address.

The keyword IS must be specified. However, this compiler allows the keyword IS to be omitted.

The location of a particular logical record must be placed in data-name before the execution of the SEEK statement (or if no SEEK statement is present, the READ and WRITE statements).

Data-name must be a fixed-length item. It must be defined in the File Section, the Working-Storage Section, or the Linkage Section. However, if data-name is specified in the File Section it may not be contained in the file for which it is the key. Data-name is made up of two components: the track identifier, and the record identifier.

TRACK IDENTIFIER: The track identifier may be expressed in two ways -- through relative track addressing, or through actual track addressing.

Relative Track Addressing: The track identifier is used to specify the relative track address at which a record is to be placed, or at which the search for a record is to begin.

Track identifier must be 4 bytes in length, and must be defined as an 8-integer binary data item whose maximum value does not exceed 16,777,215.

Actual Track Addressing: The track identifier is used to specify the actual track address at which a record is to be placed, or at which the search for a record is to begin.

Track identifier must be a binary data item eight bytes in length. No conversion is made by the compiler when determining the actual track address. The structure of these eight bytes and the permissible specifications for the 2311, 2314, and 2321 mass storage devices are shown in Figure 4.

Before beginning processing, it is the user's responsibility to initialize R to the figurative-constant LOW-VALUE. The user need not concern himself further with this field.

Device / Byte	PACK	CELL		CYLINDER		HEAD		RECORD
	M	B	B	C	C	H	H	R
Byte	0	1	2	3	4	5	6	7
2311	0-221	0	0	0	0-199	0	0-9	0-255
2314	0-221	0	0	0	0-199	0	0-19	0-255
2321	0-221	0	0-9	0-19	0-9	0-4	0-19	0-255

Figure 4. Structure of the First Eight Bytes of ACTUAL KEY -- Actual Track Addressing

RECORD IDENTIFIER: The symbolic portion of ACTUAL KEY used to identify
a particular record on a track is the record identifier.

   Record identifier must be from 1 through 255 bytes in length. Data
within these bytes is treated exactly as specified.

   A record is considered "found" when, for a given track, the record
identifier at retrieval time matches the record identifier of a record
in the file being searched.

ACTUAL KEY EXAMPLES: Two examples follow, to represent the coding
necessary to specify the data-name in the ACTUAL KEY clause.

Relative Track Addressing: The following example shows an ACTUAL KEY
using relative track addressing:

```
ENVIRONMENT DIVISION
       •
       •
       •
   ACTUAL KEY IS THE-ACTUAL-KEY.
       •
       •
       •
DATA DIVISION.
       •
       •
       •
WORKING-STORAGE SECTION.
01   THE ACTUAL-KEY.
     02   RELATIVE-TRACK-KEY USAGE COMPUTATIONAL    PICTURE IS S9(8)
          VALUE IS 10 SYNCHRONIZED.
     02   EMPLOYEE-NO   PICTURE IS X(6) VALUE IS LOW-VALUE.
```


Actual Track Addressing: The following example shows an ACTUAL KEY
using actual track addressing:

```
ENVIRONMENT DIVISION.
       •
       •
       •
ACTUAL KEY IS THE-ACTUAL-KEY.
       •
       •
       •
DATA DIVISION.
       •
       •
       •
WORKING-STORAGE SECTION.
01   BINARY-FIELD-1.
     05   TRACK-ID.
          10   M  USAGE COMPUTATIONAL    PICTURE S999 VALUE IS 0.
          10   BB USAGE COMPUTATIONAL    PICTURE S9 VALUE IS 0.
          10   CC USAGE COMPUTATIONAL    PICTURE S999 VALUE IS 10.
          10   HH USAGE COMPUTATIONAL    PICTURE S99 VALUE IS 0.
          10   R                         PICTURE IS X VALUE IS LOW-VALUE.
     05   EMPLOYEE-NO                     PICTURE XXXXXX VALUE IS LOW-VALUES.
01   ACTUAL-FIELD-1 REDEFINES BINARY-FIELD-1.
     05   FILLER                          PICTURE IS X.
     05   THE-ACTUAL-KEY                  PICTURE IS X(14).
```


   Although the track identifier field must consist of eight bytes, nine
bytes are defined within TRACK-ID. This is because the entry

        10 M USAGE COMPUTATIONAL PICTURE S999

necessarily defines two bytes. However, as Figure 4 shows, the M field
must be one byte in length. Therefore, BINARY-FIELD-1 must be redefined
as ACTUAL-FIELD-1. In this way the superfluous high-order M byte can be
stripped off from THE-ACTUAL-KEY through specification of the entry

        05 FILLER PICTURE IS X

in ACTUAL-FIELD-1. The first eight bytes of THE-ACTUAL-KEY thus
represent the track identifier, and the last six bytes represent the
record identifier.
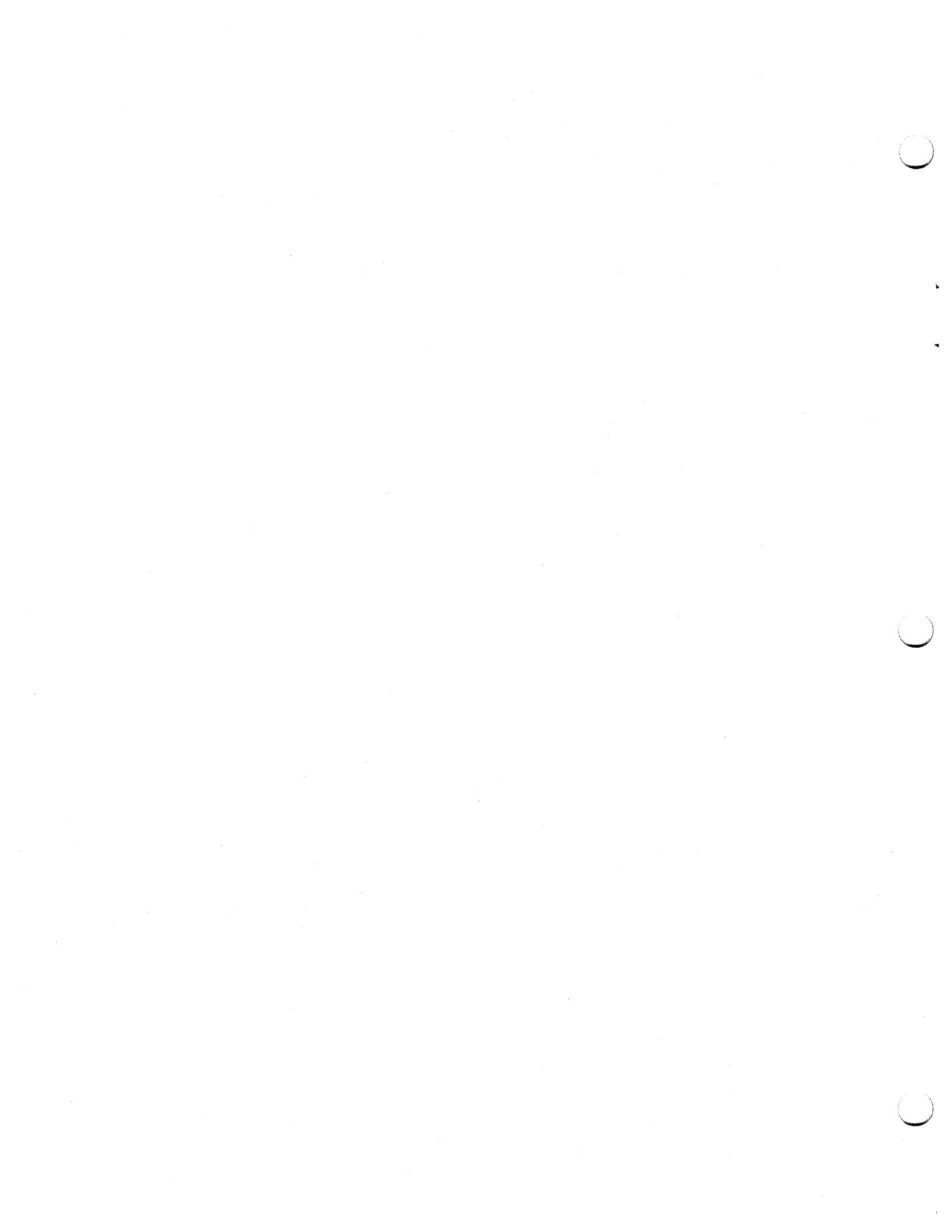
I-O-CONTROL PARAGRAPH

The I-O-CONTROL paragraph defines some of the special techniques to be used in the program. It specifies the points at which checkpoints are to be established, the core storage area which is to be shared by different files, the location of files on multiple-file reels, and optimization techniques. The I-O-CONTROL paragraph and its associated clauses are an optional part of the Environment Division.

```
r--------------------------------------------------------------------------¬
|                              General Format                              |
+--------------------------------------------------------------------------+
|                                                                          |
|      I-O-CONTROL.                                                        |
|          [RERUN Clause] ...                                             |
|          [SAME AREA Clause] ...                                         |
|          [MULTIPLE FILE TAPE Clause] ...                               |
|          [APPLY Clause] ...   .                                        |
|                                                                          |
L--------------------------------------------------------------------------+
```

RERUN Clause

The presence of a RERUN clause specifies that checkpoint records are to be taken. A checkpoint record is a recording of the status of a problem program and main storage resources at desired intervals. The contents of core storage are recorded on an external storage device at the time of the checkpoint and can be read back into core storage to restart the program from that point.

```
r--------------------------------------------------------------------------¬
|                               Format 1                                   |
+--------------------------------------------------------------------------+
|                                                                          |
|      RERUN ON system-name                                                |
|                                                                          |
|          EVERY integer RECORDS OF file-name                             |
|                                                                          |
L--------------------------------------------------------------------------+
```

```
r--------------------------------------------------------------------------¬
|                               Format 2                                   |
+--------------------------------------------------------------------------+
|                                                                          |
|      RERUN ON system-name                                                |
|                                                                          |
L--------------------------------------------------------------------------+
```

The system-name in the RERUN clause specifies the external medium for the checkpoint file, the file upon which checkpoint records are to be written. It has the following structure:

    SYSnnn-class-device-organization[-name]

The SYSnnn and name fields in the system-name for the checkpoint file cannot be the same as any specified in any ASSIGN clause.

Checkpoint records are written sequentially, and may be assigned to any utility or mass storage device (except the 2321). Only one RERUN clause in a program may use a mass storage device for writing checkpoint records. (A complete list of utility and mass storage devices is given in the description of system-name in the ASSIGN clause.)

Format 1 specifies that checkpoint records are to be written on the unit specified by system-name for every integer records of file-name that are processed. The value of integer must not exceed 16,777,215.

More than one Format 1 RERUN clause may be included in a program. If multiple RERUN clauses are specified, they may be specified either for the same or for different checkpoint files.

Format 2 is used for checkpointing sort-files and is described in "Sort."

Note: ASCII considerations for the RERUN clause are given in Appendix E.

SAME Clause

The SAME clause specifies that two or more files are to use the same core storage during processing.

```
r---------------------------------------------------------------------------
|                                    Format                                  |
|---------------------------------------------------------------------------|
|                                                                           |
|            ┌ SORT   ┐                                                      |
|    SAME    │        │   AREA FOR file-name-1 {file-name-2} ...            |
|            └ RECORD ┘                                                      |
|                                                                           |
L---------------------------------------------------------------------------
```
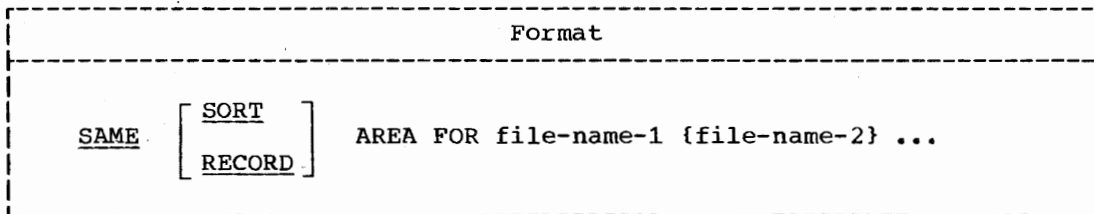
A SAME clause with the SORT option is described in "Sort." The following discussion pertains only to the SAME AREA and SAME RECORD AREA clauses.

The SAME RECORD AREA clause specifies that two or more files are to use the same main storage for processing the current logical record. All of the files may be open at the same time. A logical record in the shared storage area is considered to be:

• a logical record of each opened output file in this SAME RECORD AREA clause, and

• a logical record of the most recently read input file in this SAME RECORD AREA clause.

If the SAME clause does not contain the RECORD option, the area being shared includes all storage areas assigned to the files; therefore, it is not valid to have more than one of these files open at one time.

More than one SAME clause may be included in a program; however:

1. A _file-name_ must not appear in more than one SAME AREA clause.

2. A _file-name_ must not appear in more than one SAME RECORD AREA clause.

3. If one or more _file-names_ of a SAME AREA clause appear in a SAME RECORD AREA clause, all of the file-names in that SAME AREA clause must appear in that SAME RECORD AREA clause. However, that SAME RECORD AREA clause may contain additional file-names other than those that appear in that SAME AREA clause.

MULTIPLE FILE TAPE Clause

The MULTIPLE FILE TAPE clause is used to indicate that two or more files share the same physical reel of tape.

```
┌─────────────────────────────────────────────────────────────────────┐
│                               Format                                  │
├─────────────────────────────────────────────────────────────────────┤
│                                                                       │
│    MULTIPLE FILE TAPE CONTAINS file-name-1                            │
│                                                                       │
│        [POSITION integer-1] [file-name-2 [POSITION integer-2]] ...    │
│                                                                       │
└─────────────────────────────────────────────────────────────────────┘
```

The MULTIPLE FILE TAPE clause is required when more than one file shares the same physical reel of tape. However, this compiler permits the clause to be omitted when:

1. The files have standard labels

2. Each file is processed in the sequence in which it appears on the reel and a rewind is not executed for any file on the reel except the last file to be processed

The MULTIPLE FILE TAPE clause is pertinent only when the tape has nonstandard labels, or when labels are omitted. It is treated as comments for a tape that has standard labels.

Regardless of the number of files on a single reel, only those files that are used in the object program need be specified.

For purposes of positioning, a physical file is considered to be that segment of a tape that is terminated by a tape mark. Note that two consecutive tape marks are considered to terminate two physical files.

If all file-names refer to single physical files and have been listed in consecutive order, the POSITION option need not be given.

The POSITION integer relative to the beginning of the tape must be given if any file on the tape is not listed, or if a tape contains more than one physical file, i.e., more than one tape mark. Therefore, if a tape contains two files, each having one nonstandard header label terminated by a tape mark, their positions would be 1 and 3. If the labels are not to be processed, the positions may be specified as 2 and 4, and the LABEL RECORDS clause must specify OMITTED.

The compiler will position the tape by skipping past a number of tape marks equal to POSITION number minus one.

More than one MULTIPLE FILE clause may be included in a program.

APPLY Clause

There are six options of the APPLY clause. Any or all may appear in one COBOL program.

```
+--------------------------------------------------------------------+
|                        Format for Option 1                         |
+--------------------------------------------------------------------+
|                                                                    |
|    APPLY WRITE-ONLY ON file-name-1 [file-name-2] ...               |
|                                                                    |
+--------------------------------------------------------------------+
```

This option is used to make optimal use of buffer space allocated when creating a standard sequential file with blocked V-mode records. Normally, a buffer is truncated when the maximum size record no longer fits. Use of this option will cause a buffer to be truncated only when the next record does not fit in the buffer. The file may be opened only for OUTPUT.

The data records associated with each of the file-names must be built in a work area and written with a WRITE statement containing the FROM option. None of the subfields of these records may be referred to by procedure statements nor may any subfield be the DEPENDING ON variable in an OCCURS clause.

```
+--------------------------------------------------------------------+
|                        Format for Option 2                         |
+--------------------------------------------------------------------+
|                                                                    |
|    APPLY EXTENDED-SEARCH ON file-name-1 [file-name-2] ...          |
|                                                                    |
+--------------------------------------------------------------------+
```

This option is used to control the extent of the search made for a specified record. It may refer only to direct files opened as INPUT or I-O when ACCESS IS RANDOM. In normal operation, execution of a READ or REWRITE statement for a file causes the specified track to be searched. If the desired record cannot be found on the specified track, the INVALID KEY option of the READ or REWRITE statement is executed.

When EXTENDED-SEARCH is applied, the entire cylinder is searched for the desired record when it cannot be found on the specified track. The INVALID KEY option of the READ or REWRITE statement is executed only if the record cannot be found on the cylinder. The INVALID KEY option may also be executed when the specified track is outside the limits of the file. It is the user's responsibility to determine the cause of the INVALID KEY condition.

- DATA DIVISION -- INTRODUCTION

- ORGANIZATION OF THE DATA DIVISION

- FILE DESCRIPTION ENTRY -- DETAILS OF CLAUSES

- DATA DESCRIPTION

- DATA DESCRIPTION -- DETAILS OF CLAUSES

The Data Division of a COBOL source program contains the description
of all information to be processed by the object program. Two types of
data may be processed by a COBOL program: information recorded
externally on files and information created internally. The second
type, which exists only during the execution of a program, will be
discussed later in this chapter in "Working-Storage Section."

ORGANIZATION OF EXTERNAL DATA

A file is a collection of records. There are two types of records:
physical records and logical records. A physical record is a group of
characters or records which is treated as an entity when moved into or
out of core storage. A logical record is a number of related data
items. It may itself be a physical record, i.e., contained within a
single physical unit, or it may be one of several logical records
contained within a single physical unit, or it may extend across two or
more physical units.

COBOL source language statements provide the means of describing the
relationship between physical and logical records. Once this
relationship is established, only logical records are made available to
the COBOL programmer. Hence, in this publication, a reference to
records means logical records unless the term "physical records" is
used.

DESCRIPTION OF EXTERNAL DATA

In the discussion of data description, a distinction must first be
made between a record's external description and its internal content.

External description refers to the physical aspects of a file, i.e.,
the way in which the file appears on an external medium. For example,
the number of logical records per physical record describes the grouping
of records in the file. The physical aspects of a file are specified in
File Description entries.

A COBOL record usually consists of groups of related information that
are treated as an entity. The explicit description of the contents of
each record defines its internal characteristics. For example, the type
of data to be contained within each field of a logical record is an
internal characteristic. This type of information about each field of a
particular record is grouped into a Record Description entry.

ORGANIZATION OF THE DATA DIVISION

The Data Division is divided into four sections: the File Section, the Working-Storage Section, the Linkage Section, and the Report Section.

All data that is stored externally, for example, on magnetic tape, must be described in the File Section before it can be processed by a COBOL program. Information that is developed for internal use must be described in the Working-Storage Section. Information passed from one program to another must be described in the Linkage Section. The content and format of all reports that are to be generated by the Report Writer feature must be described in the Report Section.

The Data Division is identified by, and must begin with, the header DATA DIVISION. The File Section is identified by, and must begin with, the header FILE SECTION. The header is followed by one or more file description entries and one or more associated record description entries. The Working-Storage Section is identified by, and must begin with, the header WORKING-STORAGE SECTION. The header is followed by data item description entries for noncontiguous items, followed by record description entries. The Linkage Section is identified by, and must begin with, the header LINKAGE SECTION. The header is followed by noncontiguous data item description entries, followed by record description entries. The Report Section is identified by, and must begin with, the header REPORT SECTION. The header is followed by one or more report description entries, and one or more report group description entries.

For the proper formats of Division and Section headers, see "Use of the COBOL Coding Form" in "Language Considerations."

Structure of the Data Division

    DATA DIVISION.

    FILE SECTION.

    {file description entry

    {record description entry}...}...

    WORKING-STORAGE SECTION.

    [data item description entry]...

    [record description entry]...

    LINKAGE SECTION.

    [data item description entry]...

    [record description entry]...

    REPORT SECTION.

    {report description entry

    {report group description entry}...}...

Each of the sections of the Data Division is optional and may be omitted from the source program when the section is unnecessary. When used, the sections must appear in the foregoing sequence.

ORGANIZATION OF DATA DIVISION ENTRIES

   Each Data Division entry begins with a level indicator or a level number, followed by one or more spaces, followed by the name of a data item (except in the Report Section), followed by a sequence of independent clauses describing the data item. The last clause is always terminated by a period followed by a space.

Level Indicator

   The level indicator FD is used to specify the beginning of a file description entry. When the file is a sort-file, the level indicator SD must be used instead of FD (see "Sort"). When a report is to be generated by the Report Writer feature, the level indicator RD, specifying the beginning of a report description entry must be provided for each report in addition to the FD for the file on which the report is generated (see "Report Writer"). Figure 5 summarizes the level indicators.

Indicator	Use
FD	File description entries
SD	Sort-file description entries
RD	Report description entries

Figure 5. Level Indicator Summary

   Each level indicator must begin in Area A and be followed in Area B by its associated file-name and appropriate descriptive information.

   Level indicators are illustrated in the sample COBOL programs found in Appendix B.

Level Number

   Level numbers are used to structure a logical record to satisfy the need to specify subdivisions of a record for the purpose of data reference. Once a subdivision has been specified, it may be further subdivided to permit more detailed data reference.

   The basic subdivisions of a record, that is, those not further subdivided, are called elementary items; consequently, a record may consist of a sequence of elementary items, or the record itself may be an elementary item.

   In order to refer to a set of elementary items, the elementary items are combined into groups. A group item consists of a named sequence of one or more elementary items. Groups, in turn, may be combined into larger groups. Thus, an elementary item may belong to more than one

group. In the following example, the group items MARRIED and SINGLE are themselves part of a larger group named RETIRED-EMPLOYEES:

```
02  RETIRED-EMPLOYEES.
    03  MARRIED.
        04  NO-MALE PICTURE 9(8).
        04  NO-FEMALE PICTURE 9(8).
    03  SINGLE.
        04  NO-MALE PICTURE 9(8).
        04  NO-FEMALE PICTURE 9(8).
```

A system of level numbers shows the organization of elementary items and group items. Since records are the most inclusive data items, the level number for a record must be 1 or 01. Less inclusive data items are assigned higher (not necessarily successive) level numbers not greater than 49. There are special level numbers -- 66, 77, and 88 -- which are exceptions to this rule. Separate entries are written in the source program for each level number used.

A group includes all group and elementary items following it until a level number less than or equal to the level number of that group is encountered. The level number of an item which immediately follows the last elementary item of the previous group must be equal to the level number of one of the groups to which a prior elementary item belongs.

```
    Standard                        Nonstandard
01  A.                          01  A.
    05  C-1.                        05  C-1.
        06  D PICTURE X.                06  D PICTURE X.
        06  E PICTURE X.                06  E PICTURE X.
    05  C-2.                        04  B-1.
        .                               .
        .                               .
        .                               .
```

In the foregoing example, this compiler will accept the nonstandard use of 04 and treat it as though it had been written as an 05.

Level numbers 01 and 77 must begin in Area A, followed in Area B by associated data names and appropriate descriptive information. All other level numbers may begin in either Area A or in Area B, followed in Area B by associated data names and appropriate descriptive information.

A single-digit level number is written either as a space followed by a digit or as a zero followed by a digit. At least one space must separate a level number from the word following the level number.

Special Level Numbers

Three types of data exist whose level numbers are not intended to structure a record. They are:

66: Names of elementary items or groups described by a RENAMES clause for the purpose of regrouping data items have been assigned the special level number 66. For an example of the function of the RENAMES clause, see "Data Description."

77: Noncontiguous Working-Storage items, which are not subdivisions of other items and are not themselves subdivided, have been assigned the special level number 77.

88:   Entries that specify condition-names to be associated with particular values of a conditional variable have been assigned the special level number 88. For an example of level-88 items, see "Data Description."

Indentation

Successive data description entries may have the same format as the first such entry or may be indented according to level number. Indentation is useful for documentation purposes, and does not affect the action of the compiler.

FILE SECTION

The File Section contains a description of all externally stored data (FD), and a description of each sort-file (SD) used in the program.

The File Section must begin with the header FILE SECTION followed by a period. The File Section contains file description entries and sort-file description entries, each one followed by its associated record description entry (or entries).

```
r-------------------------------------------------------------------------------1
|                              General Format                                    |
|-------------------------------------------------------------------------------|
|                                                                               |
|   FILE SECTION.                                                               |
|                                                                               |
|   {file description entry                                                     |
|                                                                               |
|   {record description entry} ...}...                                          |
|                                                                               |
L-------------------------------------------------------------------------------J
```


File Description Entry

In a COBOL program, the File Description Entries (FD and SD) represent the highest level of organization in the File Section. The File Description entry provides information about the physical structure and identification of a file, and gives the record-name(s) associated with that file.

For a complete discussion of the sort-file-description entry, see "Sort."

Record Description Entry

The Record Description Entry consists of a set of data description entries which describe the particular record(s) contained within a particular file. For a full discussion of the format and the clauses required within the Record Description entry, see "Data Description."

WORKING-STORAGE SECTION

The Working-Storage Section may contain descriptions of records which are not part of external data files but are developed and processed internally.

The Working-Storage Section must begin with the section header WORKING-STORAGE SECTION followed by a period. The Working-Storage Section contains data description entries for noncontiguous items and record description entries, in that order.

```
+-------------------------------------------------------------------+
|                        General Format                             |
+-------------------------------------------------------------------+
|                                                                   |
|   WORKING-STORAGE SECTION.                                         |
|                                                                   |
|   [data item description entry] ...                               |
|                                                                   |
|   [record description entry] ...                                  |
|                                                                   |
+-------------------------------------------------------------------+
```

Data Item Description Entries

Noncontiguous items in Working-Storage that bear no hierarchical relationship to one another need not be grouped into records, provided they do not need to be further subdivided. Instead, they are classified and defined as noncontiguous elementary items. Each of these items is defined in a separate data item description entry that begins with the special level number 77.

Record Description Entries

Data elements in Working-Storage that bear a definite hierarchical relationship to one another must be grouped into records structured by level number.

LINKAGE SECTION

The Linkage Section describes data made available from another program (see "Subprogram Linkage" in "Procedure Division").

```
┌─────────────────────────────────────────────────────────────────────────┐
│                              General Format                               │
├───────────────────────────────────────────────────────────────────────  ┤
│                                                                           │
│   LINKAGE SECTION.                                                        │
│                                                                           │
│   [data item description entry] ...                                      │
│                                                                           │
│   [record description entry] ...                                         │
│                                                                           │
└───────────────────────────────────────────────────────────────────────  ┘
```

Data item description entries and record description entries in the
Linkage Section provide names and descriptions, but storage within the
program is not reserved since the data area exists elsewhere. Any data
description clause may be used to describe items in the Linkage Section,
with one exception: the VALUE clause may not be specified for other
than level-88 items. In the Linkage Section, the compiler assumes that
each level-01 item starts on a double-word boundary.

Note: The combined total number of level-77 and level-01 items in the
Linkage Section may not exceed 255.

REPORT SECTION

    The Report Section contains Report Description entries and report
group description entries for every report named in the REPORT clause.
The Report Section is discussed in "Report Writer."

FILE DESCRIPTION ENTRY -- DETAILS OF CLAUSES

The file description entry consists of level indicator (FD), followed by file-name, followed by a series of independent clauses. The entry itself is terminated by a period.

```
+----------------------------------------------------------------------+
|                            General Format                            |
+----------------------------------------------------------------------+
|   FD file-name                                                       |
|                                                                      |
|        [BLOCK CONTAINS Clause]                                       |
|                                                                      |
|        [RECORD CONTAINS Clause]                                      |
|                                                                      |
|        [RECORDING MODE Clause]                                       |
|                                                                      |
|         LABEL RECORDS Clause                                         |
|                                                                      |
|        [VALUE OF Clause]                                             |
|                                                                      |
|        [DATA RECORDS Clause]                                         |
|                                                                      |
|        [REPORT Clause].                                              |
|                                                                      |
+----------------------------------------------------------------------+
```

The level indicator FD identifies the beginning of a file description entry and must precede the file-name. The clauses that follow the name of the file are optional in many cases, and their order of appearance is not significant.

BLOCK CONTAINS Clause

The BLOCK CONTAINS clause is used to specify the size of a physical record.

```
+----------------------------------------------------------------------+
|                               Format                                 |
+----------------------------------------------------------------------+
|                                                    ( CHARACTERS )    |
|   BLOCK CONTAINS [integer-1 TO] integer-2          <             >    |
|                                                    (  RECORDS   )    |
|                                                                      |
+----------------------------------------------------------------------+
```

The BLOCK CONTAINS clause is unnecessary when a physical record contains one and only one complete logical record. In all other instances, this clause is required.

The BLOCK CONTAINS clause need not be specified for:

• direct files with F, U, or V mode records

• direct files when the RECORDING MODE clause is specified for S-mode records

• files containing U-mode records

For these types of files, the compiler accepts the clause and treats it as comments.

The RECORDS option may be used unless one of the following situations exists, in which case the CHARACTERS option should be used:

1. The physical record contains padding (areas not contained in a logical record)

2. Logical records are grouped in such a manner that an inaccurate physical record size would be implied. Such would be the case where the user describes a mode V record of 100 characters, yet each time he writes a block of 4, he writes a 50-character record followed by three 100-character records. Had he used the RECORDS option, the compiler would have calculated the block length as 420.

3. Logical records extend across physical records; that is, recording mode is S (spanned).

When the RECORDS option is used, the compiler assumes that the blocksize provides for integer-2 records of maximum size and then provides additional space for any required control bytes.

When the CHARACTERS option is used, the physical record size is specified in Standard Data Format, i.e., in terms of the number of bytes occupied internally by its characters, regardless of the number of characters used to represent the item within the physical record. The number of bytes occupied internally by a data item is included as part of the discussion of the USAGE clause. Integer-1 and integer-2 must include slack bytes and control bytes contained in the physical record.

When the CHARACTERS option is used, if only integer-2 is shown, it represents the exact size of the physical record. If integer-1 and integer-2 are both shown, they refer to the minimum and maximum size of the physical record, respectively.

Integer-1 and integer-2 must be positive integers.

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

When neither the CHARACTERS nor the RECORDS option is specified, the CHARACTERS option is assumed.

For an indexed file, the programmer may not use the CHARACTERS option nor the [integer-1 TO] option.

Note: ASCII considerations for the BLOCK CONTAINS clause are given in Appendix E.

RECORD CONTAINS Clause

The RECORD CONTAINS clause is used to specify the size of a file's data records.

```
r---------------------------------------------------------------------1
|                                 Format                              |
|---------------------------------------------------------------------|
|                                                                     |
|   RECORD CONTAINS [integer-1 TO] integer-2 CHARACTERS               |
|                                                                     |
L---------------------------------------------------------------------J
```

Since the size of each data record is completely defined within the record description entry, this clause is never required. When the clause is specified, the following notes apply:

1. If both integer-1 and integer-2 are shown, they refer to the number of characters in the smallest data record and the number in the largest data record, respectively.

2. Integer-2 should not be used by itself unless all the data records in the file have the same size. In this case, integer-2 represents the exact number of characters in the data record.

3. The size of the record must be specified in Standard Data Format, i.e., in terms of the number of bytes occupied internally by its characters, regardless of the number of characters used to represent the item within the record. The number of bytes occupied internally by a data item is discussed in the description of the USAGE clause. The size of a record is determined according to the rules for obtaining the size of a group item.

Normally, whether this clause is specified or omitted, the record lengths are determined by the compiler from the record descriptions. When one or more of the data item description entries within a record contains an OCCURS clause with the DEPENDING ON option, the compiler uses the maximum value of the variable to calculate the record length.

However, if more than one entry in a given record description contains an OCCURS clause with the DEPENDING ON option, and the maximum values of the variables in these OCCURS clauses do not occur simultaneously, integer-2, as specified by the user, may indicate a maximum record size other than the size calculated by the compiler from the maximum values of the OCCURS clause variables. In this case, the user-specified value of integer-2 determines the amount of storage set aside to contain the data record.

For example, in a school whose total enrollment is 500, an unblocked file of collective attendance records is being created, each record of which is described as follows:

```
    01  ATTENDANCE-RECORD.
        02   DATE PICTURE X(6).
        02   NUMBER-ABSENT PICTURE S999 USAGE IS COMP SYNC.
        02   NUMBER-PRESENT PICTURE S999 USAGE IS COMP SYNC.
        02   NAMES-OF-ABSENT OCCURS 0 TO 500 TIMES DEPENDING ON
                NUMBER-ABSENT PICTURE A(20).
        02   NAMES-OF-PRESENT OCCURS 0 TO 500 TIMES DEPENDING ON
                NUMBER-PRESENT PICTURE A(20).
```

The programmer can save storage by taking advantage of the fact that NUMBER-ABSENT plus NUMBER-PRESENT will never exceed the school's total enrollment.   Unless the programmer writes RECORD CONTAINS 10,010 CHARACTERS in the FD entry for the file, the compiler calculates the record size to be almost twice as large.

Recording Mode

When the RECORDING MODE clause is not used to specify the recording mode of the records in the file, the COBOL compiler scans each record description to determine it.   The recording mode may be F (fixed), U (undefined), V (variable), or S (spanned).

Recording Mode F -- All of the records in a file are the same length and each is wholly contained in one block. Blocks may contain more than one record, and there is usually a fixed number of records per block.   In this mode, there are no record-length or block-descriptor fields.

Recording Mode U -- The records may be either fixed or variable in length. However, there is only one record per block. There are no record-length or block-descriptor fields.

Recording Mode V -- The records may be either fixed or variable in length, and each must be wholly contained in one block. Blocks may contain more than one record. Each data record includes a record-length field and each block includes a block-descriptor field. These fields are not described in the Data Division; provision is automatically made for them. These fields are not available to the user.

Recording Mode S -- The records may be either fixed or variable in length and may be larger than a block. If a record is larger than the remaining space in a block, a segment of the record is written to fill the block. The remainder of the record is stored in the next block (or blocks if required). Only complete records are made available to the user. Each segment of a record in a block, even if it is the entire record, includes a segment-descriptor field, and each block includes a block-descriptor field. These fields are not described in the Data Division; provision is automatically made for them. These fields are not available to the user.

For standard sequential files, the compiler determines the recording mode for a given file to be:

   F if all the records are defined as being the same size and the size is smaller than or equal to the block size

   V if the records are defined as variable in size, or if the RECORD CONTAINS clause specifies variable size records and the longest record is less than or equal to the maximum block size

   S if the maximum block size is smaller than the largest record size

For direct files, the compiler determines the recording mode for a given file to be:

   F if all the records are defined as being the same size, and the size is smaller than or equal to the block size

   U if the records are defined as being variable in size, or if the RECORD CONTAINS clause specifies variable size records and the longest record is less than or equal to the maximum block size

S if the maximum block size is smaller than the largest record
   size

Files assigned to the card reader and files with indexed organization
must be F mode (fixed format).

Note: ASCII considerations for compiler calculation of recording mode
are given in Appendix E.

RECORDING MODE Clause

The RECORDING MODE clause is used to specify the format of the
logical records in the file.

```
r-----------------------------------------------------------------------------
|                                   Format                                    |
|-----------------------------------------------------------------------------|
|                                                                             |
|   RECORDING MODE IS mode                                                    |
|                                                                             |
L-----------------------------------------------------------------------------
```

Mode may be specified as F, V, U, or S. If this clause is not
specified, the recording mode is determined as described in "Recording
Mode."

The F mode (fixed-length format) may be specified when all the
logical records in a file are the same length and each is wholly
contained in one physical block. This implies that no OCCURS clause
with the DEPENDING ON option is associated with an entry in any record
description for the file. If more than one record description entry is
given following the FD entry, all record lengths calculated from the
record descriptions must be equal. Files assigned to the card reader
and files with indexed organization must be in F mode.

The V mode (variable-length format) may be specified for any
combination of record descriptions if each record is wholly contained
within one physical block. A mode V record is preceded by a control
field containing the length of the logical record. Blocks of
variable-length records include a block-descriptor control field. V
mode may be specified only for standard sequential files.

The U mode (undefined format) may be specified for any combination of
record descriptions if each record is wholly contained within one
physical block. It is comparable to V mode with the exception that U
mode records are not blocked and have no preceding control field. U
mode may be specified only for direct files or standard sequential
files.

The S mode (spanned format) may be specified for any combination of
record descriptions. A record that cannot fit into the remaining space
in a block appears as multiple segments, one segment per block. A
record that can be completely contained in a block appears as a single
segment. An S-mode segment is preceded by a control field containing
the length of the segment and indicating whether it is the first and/or
last or an intermediate segment. Blocks of S-mode segments include a
block-descriptor control field. S mode may be specified for standard
sequential or direct files.

Note: ASCII considerations for the RECORDING MODE clause are given in
Appendix E.

LABEL RECORDS Clause

The LABEL RECORDS clause specifies whether labels are present, and if present, identifies the labels.

```
+-----------------------------------------------------------------------+
|                               Format                                  |
+-----------------------------------------------------------------------+
|                                                                       |
|        (RECORD IS  )   (OMITTED                        )              |
| LABEL  {            )   {STANDARD                       }              |
|        (RECORDS ARE)   (data-name-1 [data-name-2] ...  )              |
|                                                                       |
+-----------------------------------------------------------------------+
```

The LABEL RECORDS clause is required in every FD.

The OMITTED option specifies either that no explicit labels exist for the file or that the existing labels are nonstandard and the user does not want them to be processed by a label declarative (i.e., they will be processed as data records). The OMITTED option must be specified for files assigned to unit record devices. It may be specified for files assigned to magnetic tape units. Use of the OMITTED option does not result in automatic bypassing of nonstandard labels on input. It is the user's responsibility either to process or to bypass nonstandard labels on input and create them on output.

STANDARD specifies that labels exist for the file and the labels conform to system specification. The system will bypass user labels appearing in the file if the STANDARD option is specified.

The STANDARD option must be specified for files with indexed organization.

In the discussion that follows, all references to data-name-1 apply equally to data-name-2.

The data-name-1 option indicates either the presence of user labels in addition to standard labels, or the presence of nonstandard labels. Data-name-1 specifies the name of a user label record. Data-name-1 must appear as the subject of a record description entry associated with the file, and must not appear as an operand of the DATA RECORDS clause for the file.

If user labels are to be processed, data-name-1 may be specified for direct files, or for standard sequential files with the exception of files assigned to unit-record devices.

A user label is 80 characters in length. A user header label must have UHL in character positions 1 through 3. A user trailer label must have UTL in character positions 1 through 3. Both header and trailer labels may be grouped and each label must show the relative position (1, 2, ...) of the label within the user label group, in character position 4. The remaining 76 characters are formatted according to the user's choice. User header labels follow standard beginning file labels but precede the first data record; user trailer labels follow standard closing file labels.

If nonstandard labels are to be processed, data-name-1 may be specified only for standard sequential files, with the exception of files assigned to unit-record devices. The length of a nonstandard label may not exceed 4,095 character positions.

All Procedure Division references to data-name-1, or to any item subordinate to data-name-1, must appear within label processing declaratives.

Note: ASCII considerations for the LABEL RECORDS clause are given in Appendix E.

VALUE OF Clause

The VALUE OF clause particularizes the description of an item in the label records associated with a file, and serves only as documentation.

```
r----------------------------------------------------------------------1
|                              Format                                   |
|----------------------------------------------------------------------|
|                                                                       |
|                                  (data-name-2)                        |
|    VALUE OF data-name-1 IS       {            }                       |
|                                  (literal-1  )                        |
|                                                                       |
|                             (data-name-4)                             |
|         [data-name-3 IS     {           }  ] ...                      |
|                             (literal-2  )                             |
|                                                                       |
L----------------------------------------------------------------------J
```

To specify the required values of identifying data items in the label records for the file, the programmer must use the VALUE OF clause.

However, this compiler treats the VALUE OF clause as comments, since for standard labels this function is performed by the system through the TLBL or DLBL control statement as described in the Programmer's Guides (as cited in "Preface"), and through the Label Declarative procedures for user standard labels and nonstandard labels.

DATA RECORDS Clause

The DATA RECORDS clause serves only as documentation, and identifies the records in the file by name.

```
r----------------------------------------------------------------------1
|                              Format                                   |
|----------------------------------------------------------------------|
|                                                                       |
|         (RECORD IS   )                                                |
|    DATA  {           }   data-name-1 [data-name-2] ...                |
|         (RECORDS ARE )                                                |
|                                                                       |
L----------------------------------------------------------------------J
```

The presence of more than one data-name indicates that the file contains more than one type of data record. That is, two or more record descriptions for a given file occupy the same storage area. These records need not have the same description. The order in which the data-names are listed is not significant.

Data-name-1, data-name-2, etc., are the names of data records and each must be preceded in its record description entry by the level number 01.

This clause is never required.

REPORT Clause

The REPORT clause is used in conjunction with the Report Writer feature. A complete description of the REPORT clause can be found in "Report Writer."
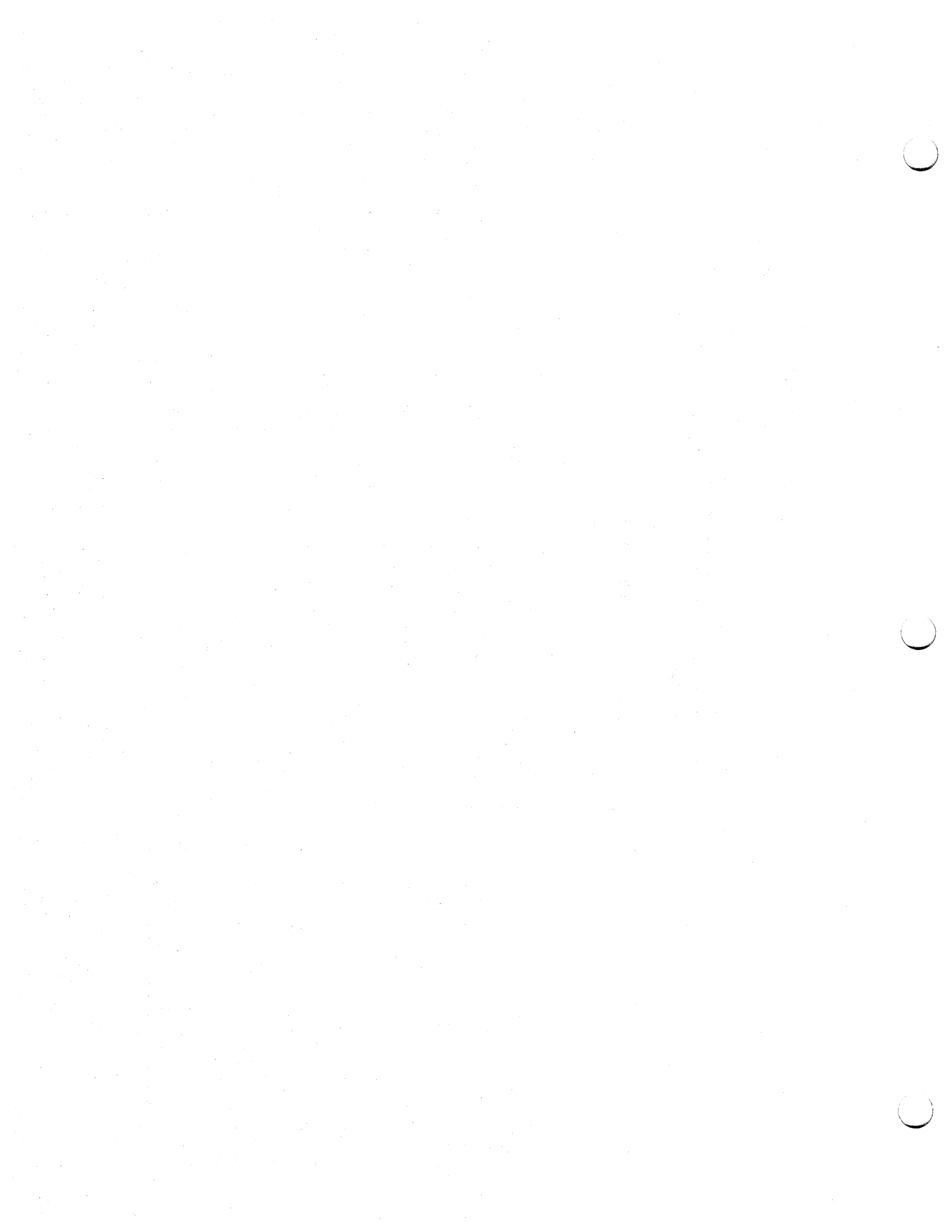
In COBOL, the terms used in connection with data description are:

Data Description Entry -- the clause, or clauses, that specify the characteristics of any particular noncontiguous data item, or of any data item that is a portion of a record. The data description entry consists of a level number, a data-name (or condition-name), plus any associated data description clauses.

Data Item Description Entry -- a data description entry that defines a noncontiguous data item. It consists of a level number (77), a data-name plus any associated data description entries. Data item description entries are valid in the Working-Storage Section, and in the Linkage Section.

Record Description Entry -- the term used in connection with a record. It consists of a hierarchy of data description entries. Record description entries are valid in the File, Working-Storage, and Linkage Sections.

The maximum length for a data description entry is 32,767 bytes, except for a fixed-length Working-Storage or Linkage Section group item, which may be as long as 131,071 bytes.

```
+---------------------------------------------------------------------+
|                          General Format 1                           |
+---------------------------------------------------------------------+
|                                                                     |
|                           ( data-name )                             |
|     level number          {           }                             |
|                           ( FILLER    )                             |
|                                                                     |
|        [REDEFINES Clause]                                           |
|        [BLANK WHEN ZERO Clause]                                     |
|        [JUSTIFIED Clause]                                           |
|        [OCCURS Clause]                                              |
|        [PICTURE Clause]                                             |
|        [SYNCHRONIZED Clause]                                        |
|        [USAGE Clause]                                               |
|        [VALUE Clause].                                              |
|                                                                     |
+---------------------------------------------------------------------+
```

```
+---------------------------------------------------------------------+
|                          General Format 2                           |
+---------------------------------------------------------------------+
|                                                                     |
|     66 data-name-1 RENAMES Clause.                                 |
|                                                                     |
+---------------------------------------------------------------------+
```

```
+---------------------------------------------------------------------+
|                          General Format 3                           |
+---------------------------------------------------------------------+
|                                                                     |
|     88 condition-name VALUE Clause.                                |
|                                                                     |
+---------------------------------------------------------------------+
```

General Format 1 is used for record description entries in the File, Working-Storage, and Linkage Sections and for data item description entries in the Working-Storage and Linkage Sections. The following rules apply:

1. Level number may be any number from 1 through 49 for record description entries, or 77 for data item description entries.

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

3. The PICTURE clause must be specified for every elementary item, with the exception of index data items and internal floating-point items. Index data items are described in "Table Handling."

4. Each entry must be terminated by a period.

5. Semicolons or commas may be used as separators between clauses.

General Format 2 is used for the purpose of regrouping data items. The following rules apply:

1. A level-66 entry cannot rename another level-66 entry, nor can it rename a level-77, level-88, or level-01 entry.

2. All level-66 entries associated with a given logical record must immediately follow the last data description entry in the record.

3. The entry must be terminated by a period.

   The RENAMES clause is discussed in detail later in this chapter.

General Format 3 is used to describe entries that specify condition-names to be associated with particular values of a conditional variable. A condition-name is a name assigned by the user to a specific value that a data item may assume during object program execution. The following rules apply:

1. The condition-name entries for a particular conditional variable must immediately follow the conditional variable.

2. A condition-name can be associated with any data description entry except another condition-name, a level-66 item, an index data item, or a group containing items with descriptions including JUSTIFIED, SYNCHRONIZED, or USAGE (other than USAGE IS DISPLAY).

3. The entry must be terminated by a period.

In the following example, PAYROLL-PERIOD is the conditional variable. WEEKLY, SEMI-MONTHLY, and MONTHLY are the condition-names associated with it. The PICTURE associated with PAYROLL-PERIOD limits the value of each condition-name to one digit.

```
02  PAYROLL-PERIOD PICTURE IS 9.
    88 WEEKLY VALUE IS 1.
    88 SEMI-MONTHLY VALUE IS 2.
    88 MONTHLY VALUE IS 3.
```

The data description entry consists of a level number, followed by a data-name, followed by a series of independent clauses. The clauses may be written in any order, with one exception: the REDEFINES clause, when used, must immediately follow the data-name. The entry must be terminated by a period.

Data-name or FILLER Clause

A data-name specifies the name of the data being described. The word FILLER specifies an elementary or group item of the logical record that is never referred to and therefore need not be named.

```
+------------------------------------------------------------------+
|                             Format                               |
+------------------------------------------------------------------+
|                                                                  |
|                         ( data-name )                            |
|         level number   {            }                            |
|                         ( FILLER    )                            |
|                                                                  |
+------------------------------------------------------------------+
```
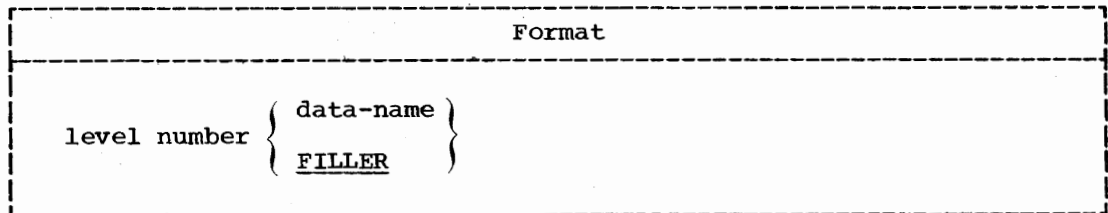
In the Working-Storage, Linkage, or File Sections, a data-name or the key word FILLER must be the first word following the level number in each data description entry.

A data-name is a name assigned by the user to identify a data item used in a program. A data-name refers to a kind of data, not to a particular value; the item referred to may assume a number of different values during the course of a program.

The key word FILLER is used to specify an elementary item or a group item that is never referred to in the program, and therefore need not be named. Under no circumstances may a FILLER item be referred to directly. In a MOVE, ADD, or SUBTRACT statement with the CORRESPONDING option, FILLER items are ignored.

Note: Level-77 and level-01 entries in the Working-Storage and Linkage Section must be given unique data-names, since neither can be qualified. Subordinate data-names, if they can be qualified, need not be unique.

REDEFINES Clause

The REDEFINES clause allows the same computer storage area to contain different data items or provides an alternative grouping or description of the same data. That is, the REDEFINES clause specifies the redefinition of a storage area, not of the data items occupying the area.

```
+-----------------------------------------------------------------------+
|                               Format                                  |
+-----------------------------------------------------------------------+
|                                                                       |
|    level number data-name-1 REDEFINES data-name-2                     |
|                                                                       |
+-----------------------------------------------------------------------+
```

The level numbers of data-name-1 and data-name-2 must be identical, but must not be 66 or 88. Data-name-2 is the name associated with the previous data description entry. Data-name-1 is an alternate name for the same area. When written, the REDEFINES clause must be the first clause following data-name-1.

The REDEFINES clause must not be used in level-01 entries in the File Section. Implicit redefinition is provided when more than one level-01 entry follows a file description entry.

Redefinition starts at data-name-2 and ends when a level number less than or equal to that of data-name-2 is encountered. Between the data descriptions of data-name-2 and data-name-1, there may be no entries having lower level numbers (numerically) than the level number of data-name-2 and data-name-1. Example:

```
02  A.
    03  A-1         PICTURE X.
    03  A-2         PICTURE XXX.
    03  A-3         PICTURE 99.
02  B REDEFINES A PICTURE X(6).
```

In this case, B is data-name-1, and A is data-name-2. When B redefines A, the redefinition includes all of the items subordinate to A (A-1, A-2, and A-3).

The data description entry for data-name-2 cannot contain an OCCURS clause, nor can data-name-2 be subordinate to an entry which contains an OCCURS clause. An item subordinate to data-name-2 may contain an OCCURS clause without the DEPENDING ON option. Data-name-1 or any items subordinate to data-name-1 may contain an OCCURS clause without the DEPENDING ON option. Neither data-name-2 nor data-name-1 nor any of their subordinate items may contain an OCCURS clause with the DEPENDING ON option. When data-name-1 has a level number other than 01, it must specify a storage area of the same size as data-name-2.

However, this compiler will allow the size of the redefining area (data-name-1) to be less than the size of the redefined area (data-name-2).

If data-name-1 contains an OCCURS clause, its size is computed by multiplying the length of one occurrence by the number of occurrences.

Note: In the discussion that follows, the term "computational" refers to COMPUTATIONAL, COMPUTATIONAL-1, and COMPUTATIONAL-2 items.

When the SYNCHRONIZED clause is specified for an item that also contains a REDEFINES clause, the data item that is redefined must have the proper boundary alignment for the data item that REDEFINES it. For example, if the programmer writes:

```
02  A              PICTURE X(4).
02  B  REDEFINES A  PICTURE S9(9) COMP SYNC.
```

he must ensure that A begins on a fullword boundary.

When the SYNCHRONIZED clause is specified for a computational item that is subordinate to an item that contains a REDEFINES clause, the computational item must not require the addition of slack bytes.

Except for condition-name entries, the entries giving the new description of the storage area must not contain any VALUE clauses.

The entries giving the new description of the storage area must follow the entries describing the area being redefined, without intervening entries that define new storage areas. Multiple redefinitions of the same storage area should all use the data-name of the entry that originally defined the area. However, this compiler will accept as valid the data-name of the preceding entry when multiple redefinition is used. For example, both of the following are valid uses of the REDEFINES clause:

```
02  A            PICTURE 9999.
02  B REDEFINES A PICTURE 9V999.
02  C REDEFINES A PICTURE 99V99.

02  A            PICTURE 9999.
02  B REDEFINES A PICTURE 9V999.
02  C REDEFINES B PICTURE 99V99.
```

Data items within an area can be redefined without their lengths being changed; the following statements result in the storage layout shown in Figure 6.

```
02  NAME-2.
    03  SALARY    PICTURE XXX.
    03  SO-SEC-NO PICTURE X(9).
    03  MONTH     PICTURE XX.
02  NAME-1 REDEFINES NAME-2.
    03  WAGE      PICTURE XXX.
    03  MAN-NO    PICTURE X(9).
    03  YEAR      PICTURE XX.
```
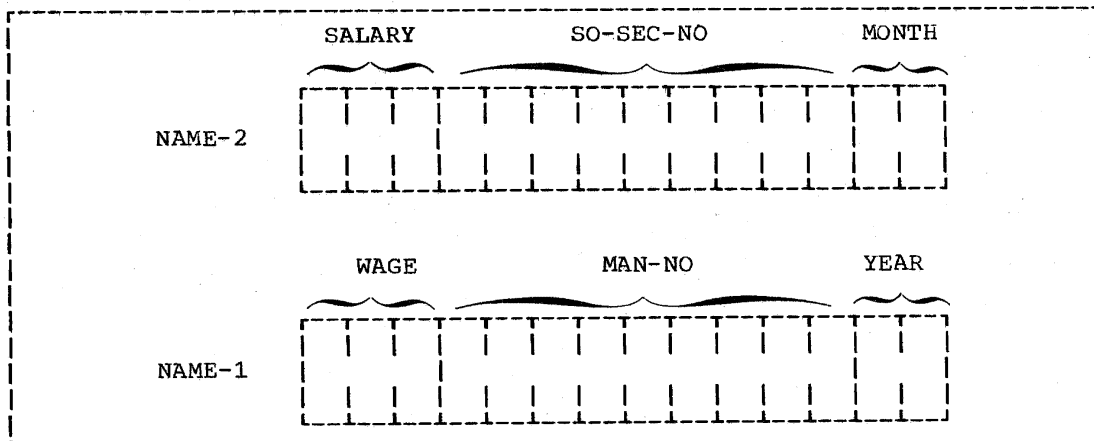

Figure 6. Areas REDEFINED without Changes in Length

Data items can also be rearranged within an area; the following statements result in the storage layout shown in Figure 7.

```
02  NAME-2.
    03  SALARY      PICTURE XXX.
    03  SO-SEC-NO   PICTURE X(9).
    03  MONTH       PICTURE XX.
02  NAME-1  REDEFINES  NAME-2.
    03  MAN-NO      PICTURE X(6).
    03  WAGE        PICTURE 999V999.
    03  YEAR        PICTURE XX.
```
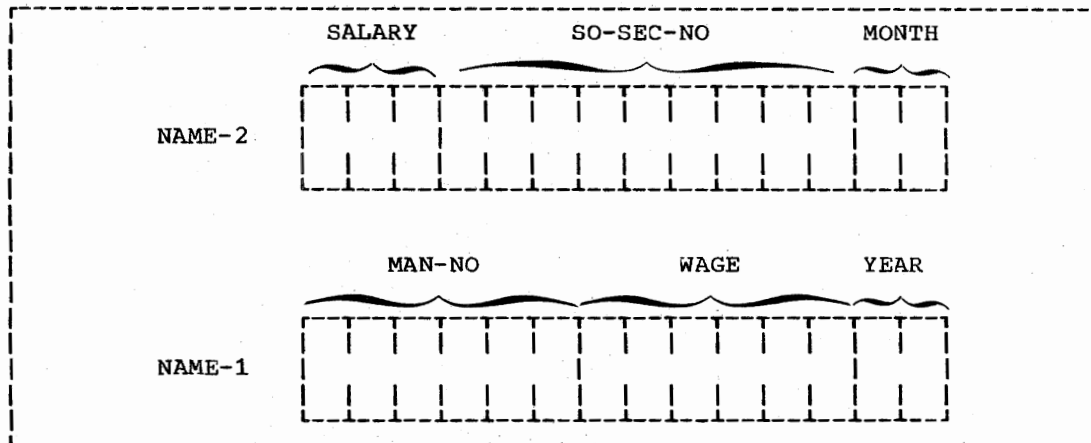

Figure 7. Areas REDEFINED and Rearranged

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

The usage of data items within an area can be redefined.

Altering the USAGE of an area through redefinition does not cause any change in existing data. Consider the example:

```
02  B              PICTURE 99 USAGE DISPLAY  VALUE IS 8.
02  C  REDEFINES B  PICTURE S99 USAGE COMPUTATIONAL.
02  A              PICTURE S9999 USAGE COMPUTATIONAL.
```

Assuming that B is on a halfword boundary, the bit configuration of the value 8 is 1111 0000 1111 1000, because B is a DISPLAY item. Redefining B does not change its appearance in storage. Therefore, a great difference results from the two statements ADD B TO A and ADD C TO A. In the former case, the value 8 is added to A, because B is a display item. In the latter case, the value -3,848 is added to A, because C is a binary item (USAGE IS COMPUTATIONAL), and the bit configuration appears as a negative number.

Moving a data item to a second data item that redefines the first one (for example, MOVE B TO C when C redefines B), may produce results that are not those expected by the programmer.   The same is true of the reverse (MOVE B TO C when B redefines C).

A REDEFINES clause may be specified for an item within the scope of an area being redefined, that is, an item subordinate to a redefined item.   The following example would thus be a valid use of the REDEFINES clause:

```
02   REGULAR-EMPLOYEE.
     03   LOCATION                                    PICTURE A(8).
     03   STATUS                                      PICTURE X(4).
     03   SEMI-MONTHLY-PAY                            PICTURE 9999V99.
     03   WEEKLY-PAY   REDEFINES SEMI-MONTHLY-PAY PICTURE 999V999.
02   TEMPORARY-EMPLOYEE REDEFINES REGULAR-EMPLOYEE.
     03   LOCATION                                    PICTURE A(8).
     03   FILLER                                      PICTURE X(6).
     03   HOURLY-PAY                                  PICTURE 99V99.
```

REDEFINES clauses may also be specified for items subordinate to items containing REDEFINES clauses.   For example:

```
02   REGULAR-EMPLOYEE.
     03   LOCATION                                    PICTURE A(8).
     03   STATUS                                      PICTURE X(4).
     03   SEMI-MONTHLY-PAY                            PICTURE 999V999.
02   TEMPORARY-EMPLOYEE   REDEFINES REGULAR-EMPLOYEE.
     03   LOCATION                                    PICTURE A(8).
     03   FILLER                                      PICTURE X(6).
     03   HOURLY-PAY                                  PICTURE 99V99.
     03   CODE-H   REDEFINES   HOURLY-PAY             PICTURE 9999.
```

BLANK WHEN ZERO Clause

This clause specifies that an item is to be set to blanks whenever its value is zero.

```
+-----------------------------------------------------------------------+
|                              Format                                   |
+-----------------------------------------------------------------------+
|                                                                       |
|   BLANK WHEN ZERO                                                     |
|                                                                       |
+-----------------------------------------------------------------------+
```

When the BLANK WHEN ZERO clause is used, the item will contain only blanks if the value of the item is zero.

The BLANK WHEN ZERO clause may be specified only at the elementary level for numeric edited or numeric items.   When this clause is used for an item whose PICTURE is numeric, the category of the item is considered to be numeric edited.

This clause may not be specified for level-66 and level-88 data items.

JUSTIFIED Clause

The JUSTIFIED clause is used to override normal positioning of data within a receiving alphabetic or alphanumeric data item.

```
+-------------------------------------------------------------------+
|                              Format                               |
+-------------------------------------------------------------------+
|                                                                   |
|  ( JUSTIFIED )                                                    |
|  {          }   RIGHT                                             |
|  ( JUST     )                                                     |
|                                                                   |
+-------------------------------------------------------------------+
```

Normally, the rule for positioning data within a receiving alphanumeric or alphabetic data item is:

* The data is aligned in the receiving field, beginning at the leftmost character position within the receiving field. Unused character positions to the right are filled with spaces. If truncation occurs, it will be at the right.

The JUSTIFIED clause affects the positioning of data in the receiving field as follows:

* When the receiving data item is described with the JUSTIFIED clause and the data item sent is larger than the receiving data item, the leftmost characters are truncated.

* When the receiving data item is described with the JUSTIFIED clause and is larger than the data item sent, the data is aligned at the rightmost character position in the data item. Unused character positions to the left are filled with spaces.

The JUSTIFIED clause may only be specified for elementary items.

This clause must not be specified for level-66 or level-88 data items.

OCCURS Clause

The OCCURS clause is used to define tables and other homogeneous sets of data, whose elements can be referred to by subscripting or indexing. The OCCURS clause is described in "Table Handling."

PICTURE Clause

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

```
r----------------------------------------------------------------------¬
|                              Format                                   |
+----------------------------------------------------------------------+
|                                                                      |
| ⎧ PICTURE ⎫                                                           |
| ⎨         ⎬    IS character string                                   |
| ⎩ PIC     ⎭                                                           |
|                                                                      |
L----------------------------------------------------------------------J
```
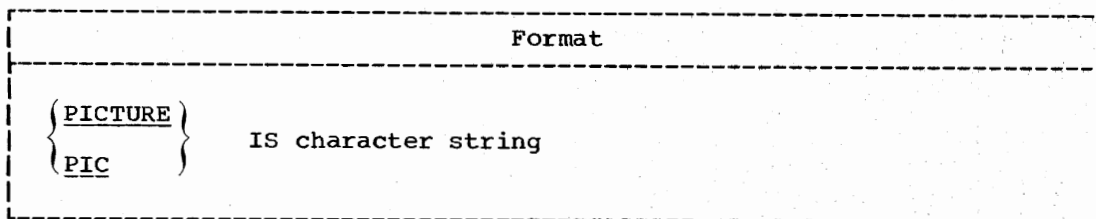
The PICTURE clause can be used only at the elementary level.

The <u>character string</u> consists of certain allowable combinations of characters in the COBOL character set. The maximum number of characters allowed in the character string is 30. The allowable combinations determine the category of the elementary item.

There are five categories of data that can be described with a PICTURE clause. They are:

1. Alphabetic
2. Numeric
3. Alphanumeric
4. Alphanumeric edited
5. Numeric edited

The Three Classes of Data

The five categories of data items are grouped into three classes: alphabetic, numeric, and alphanumeric. For alphabetic and numeric, the class and the category are synonymous. The alphanumeric class includes the categories of alphanumeric (without editing), alphanumeric edited, and numeric edited.

Every elementary item belongs to one of the three classes and to one of the five categories. The class of a group item is treated at object time as alphanumeric regardless of the class of the elementary items subordinate to that group item.

Table 6 shows the relationship of the class and category for elementary and group data items.

Table 6. Class and Category of Elementary and Group Data Items

Level of Item	Class	Category
Elementary	Alphabetic	Alphabetic
	Numeric	Numeric
	Alphanumeric	Alphanumeric Alphanumeric Edited Numeric Edited
Group	Alphanumeric	Alphabetic Numeric Alphanumeric Alphanumeric Edited Numeric Edited

Character String and Item Size

In the processing of data through COBOL statements, the size of an elementary item is determined through the number of character positions specified in its PICTURE character string. In core storage, however, the size is determined by the actual number of bytes the item occupies, as determined by its PICTURE character string, and also by its USAGE (see "USAGE Clause").

Normally, when an arithmetic item is moved from a longer field into a shorter one, this compiler will truncate the data to the number of characters represented in the PICTURE character string of the shorter item.

For example, if a sending field with PICTURE S99999, and containing the value +12345, is moved to a COMPUTATIONAL receiving field with PICTURE S99, the data is truncated to +45.

As a compile time option, however, this compiler may be instructed, in such an operation, to truncate only such digits as would overflow the receiving field. If this option is used, the result of the move in the foregoing example is +2345, since a COMPUTATIONAL item two bytes in length can contain up to four decimal digits of data. Note that care must be used when using this option, since there are times when the data may contain a negative sign.

Repetition of Symbols

An integer which is enclosed in parentheses following one of the symbols

    A   ,   X   9   P   Z   *   B   0   +   − $

indicates the number of consecutive occurrences of the symbol. For example, if the programmer writes

    A(40)

the four characters (40) indicate forty consecutive appearances of the symbol A. The number within parentheses may not exceed 32,767.

Note: The following symbols may appear only once in a given PICTURE clause:

    S   V   .   CR   DB

Symbols Used in the PICTURE Clause

The functions of the symbols used to describe an elementary item are:

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

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

P   The P indicates an assumed decimal scaling position and is used to specify the location of an assumed decimal point when the point is not within the number that appears in the data item. The scaling position character P is not counted in the size of the data item. Scaling position characters are counted in determining the maximum

number of digit positions (18) in numeric edited items or in items
that appear as operands in arithmetic statements.

The scaling position character P may appear only to the left or
right of the other characters in the string as a continuous string
of P's within a PICTURE description. The sign character S and the
assumed decimal point V are the only characters which may appear to
the left of a leftmost string of P's. Since the scaling position
character P implies an assumed decimal point (to the left of the
P's if the P's are leftmost PICTURE characters and to the right of
the P's if the P's are rightmost PICTURE characters), the assumed
decimal point symbol V is redundant as either the leftmost or
rightmost character within such a PICTURE description.

S   The symbol S is used in a PICTURE character string to indicate the
presence (but not the representation nor, necessarily, the
position) of an operational sign, and must be written as the
leftmost character in the PICTURE string. An operational sign
indicates whether the value of an item involved in an operation is
positive or negative. The symbol S is not counted in determining
the size of the elementary item, unless an associated SIGN clause
specifies the SEPARATE CHARACTER option.

V   The V is used in a character string to indicate the location of the
assumed decimal point and may appear only once in a character
string. The V does not represent a character position and,
therefore, is not counted in the size of the elementary item. When
the assumed decimal point is to the right of the rightmost symbol
in the string, the V is redundant.

X   Each X in the character string represents a character position
which may contain any allowable character from the EBCDIC set.

Z   Each Z in the character string represents a leading numeric
character position; when that position contains a zero, the zero is
replaced by a space character. Each Z is counted in the size of
the item.

9   Each 9 in the character string represents a character position that
contains a numeral and is counted in the size of the item.

0   Each zero in the character string represents a character position
into which the numeral zero will be inserted. The 0 is counted in
the size of the item.

,   Each comma in the character string represents a character position
into which a comma will be inserted. This character is counted in
the size of the item. The comma insertion character cannot be the
last character in the PICTURE character string.

.   When a period appears in the character string, it is an editing
symbol that represents the decimal point for alignment purposes.
In addition, it represents a character position into which a period
will be inserted. This character is counted in the size of the
item. The period insertion character cannot be the last character
in the PICTURE character string.

Note: For a given program, the functions of the period and comma
are exchanged if the clause DECIMAL-POINT IS COMMA is stated in the
SPECIAL-NAMES paragraph. In this exchange, the rules for the
period apply to the comma and the rules for the comma apply to the
period wherever they appear in a PICTURE clause.

±   These symbols are used as editing sign control symbols. When
CR   used, each represents the character position into which the
DB   editing sign control symbol will be placed. The symbols are

mutually exclusive in one character string. Each character used in the symbol is counted in determining the size of the data item.

* Each asterisk (check protect symbol) in the character string represents a leading numeric character position into which an asterisk will be placed when that position contains a zero. Each * is counted in the size of the item.

$ The currency symbol in the character string represents a character position into which a currency symbol is to be placed. The currency symbol in a character string is represented either by the symbol $ or by the single character specified in the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph of the Environment Division. The currency symbol is counted in the size of the item.

Table 7 shows the order of precedence of the symbols used in the PICTURE clause.

Table 7. Precedence of Symbols Used in the PICTURE Clause

SECOND SYMBOL \ FIRST SYMBOL		NON-FLOATING INSERTION SYMBOLS B	0	,	.	{+ −}	{+ −}	{CR DB}	cs¹	FLOATING INSERTION SYMBOLS {Z *}	{Z *}	{+ −}	{+ −}	cs¹	cs¹	OTHER SYMBOLS 9	A/X	S	V	P	P
NON-FLOATING INSERTION SYMBOLS	B	X	X	X	X	X			X	X	X	X	X	X	X	X	X		X		X
	0	X	X	X	X	X			X	X	X	X	X	X	X	X	X		X		X
	,	X	X	X	X	X			X	X	X	X	X	X	X	X			X		X
	.	X	X	X		X			X	X		X		X		X					
	+ or −																				
	+ or −	X	X	X	X				X	X	X			X	X	X			X	X	X
	CR or DB	X	X	X	X				X	X	X			X	X	X			X	X	X
	cs¹					X															
FLOATING INSERTION SYMBOLS	Z or *	X	X	X		X			X	X											
	Z or *	X	X	X	X	X			X	X	X								X		X
	+ or −	X	X	X					X			X									
	+ or −	X	X	X	X				X			X	X						X		X
	cs¹	X	X	X		X								X							
	cs¹	X	X	X	X	X								X	X				X		X
OTHER SYMBOLS	9	X	X	X	X	X			X	X		X		X		X	X	X	X		X
	A X	X	X														X	X				
	S																				
	V	X	X	X		X			X	X		X		X		X			X		X
	P	X	X	X		X			X	X		X		X		X			X		X
	P					X			X										X	X	X

¹cs is the abbreviation for the currency symbol.

At least one of the symbols A, X, Z, 9, or *, or at least two of the symbols +, −, or cs must be present in a PICTURE string.

An X at an intersection indicates that the symbol(s) at the top of the column may, in a given character-string, appear anywhere to the left of the symbol(s) at the left of the row.

Non-floating insertion symbols + and −, floating insertion symbols Z, *, +, −, and cs, and other symbol P appear twice in the above PICTURE character precedence table. The leftmost column and uppermost row for each symbol represents its use to the left of the decimal point position. The second appearance of the symbol in the table represents its use to the right of the decimal point position.

Braces ({}) indicate items that are mutually exclusive

The Five Categories of Data

The following is a detailed description of the allowable combinations of characters for each category of data.

ALPHABETIC ITEMS: An alphabetic item is one whose PICTURE character string contains only the symbol A. Its contents, when represented in Standard Data Format, must be any combination of the 26 letters of the Roman alphabet and the space from the COBOL character set. Each alphabetic character is stored in a separate byte.

ALPHANUMERIC ITEMS: An alphanumeric item is one whose PICTURE character string is restricted to combinations of the symbols A, X, and 9. The item is treated as if the character string contained all X's. Its contents, when represented in Standard Data Format, are allowable characters from the EBCDIC set.

A PICTURE character string which contains all A's or all 9's does not define an alphanumeric item.

NUMERIC ITEMS: There are two types of numeric items: fixed-point items and floating-point items.

Fixed-Point Numeric Items: There are three types of fixed-point numeric items: external decimal, binary, and internal decimal. See the discussion of the USAGE clause for details concerning each.

The PICTURE of a fixed-point numeric item may contain a valid combination of the following symbols:

    9   V   P   S

Examples of fixed-point numeric items:

PICTURE	Valid Range of Values
9999	0 through 9999
S99	-99 through +99
S999V9	-999.9 through +999.9
PPP999	0 through .000999
S999PPP	-1000 through -999000 and
	+1000 through +999000 or zero

The maximum size of a fixed-point numeric item is 18 digits.

The contents of a fixed-point numeric item, when represented in Standard Data Format, must be a combination of the Arabic numerals 0 through 9; the item may contain an operational sign. If the PICTURE contains an S, the contents of the item are treated as positive or negative values, depending on the operational sign; if the PICTURE does not contain an S, the contents of the item are treated as absolute values.

Note: ASCII considerations for the PICTURE clause are given in Appendix E.

Floating-Point Numeric Items: These items define data whose potential range of value is too great for fixed-point presentation. The magnitude of the number represented by a floating-point item must be greater than 5.4×10^{-79} but must not exceed $.72 \times 10^{76}$.

There are two types of floating-point items: internal floating-point and external floating-point. See the discussion of the USAGE clause for details concerning each.

No PICTURE clause may be associated with an internal floating-point item.

An <u>external floating-point</u> item has a PICTURE character string in the following form:

  {±}mantissa <u>E</u>{±}exponent

where each element of the string is composed according to the following rules:

{±}   A plus sign or a minus sign must immediately precede both the mantissa and the exponent in the PICTURE character string.

+   indicates that a plus sign in the data represents positive values and a minus sign represents negative values.

-     indicates that a space character in the data represents positive values and that a minus sign represents negative values.

      The plus sign, the space character, and the minus sign occupy a byte of storage, and are included in the printout.

<u>mantissa</u> The mantissa immediately follows the first sign character, and is represented using the following three symbols:

9   Each <u>9</u> in the mantissa character string represents a digit position into which a numeric character will be placed. From one to sixteen 9's may be present in the string. Each digit position occupies a byte of storage.

.   indicates an actual decimal point. It occupies a byte of storage.

V   indicates an assumed decimal point. It does not take up any storage.

      One actual or assumed decimal point must be present in the mantissa as a leading, embedded or trailing symbol.

E   indicates the exponent, and immediately follows the mantissa. It occupies one byte of storage.

<u>exponent</u> The exponent immediately follows the second sign character. It is represented by two consecutive 9's. Each occupies a byte of storage.

   External data must conform to the representation specified in the PICTURE clause.

Examples of external floating-point items:

PICTURE	Format of External Data	Value Expressed
-9V99E-99	540E-79	$+5.40 \times 10^{-79}$
+999.99E+99	+123.45E-14	$+123.45 \times 10^{-14}$
-V9(6)E+99	565656E+45	$+.565656 \times 10^{45}$
+.9(10)E-99	+.7200000000E 76	$+.72 \times 10^{76}$

(Note that any of the above PICTURE representations can express the full range of possible values.)

   No VALUE clause may be associated with an external floating-point item.

PICTURE Clause

ALPHANUMERIC EDITED ITEMS: An alphanumeric edited item is one whose PICTURE character string is restricted to certain combinations of the following symbols:

    A   X   9   B   0

To qualify as an alphanumeric edited item, one of the following conditions must be true:

1. The character string must contain at least one B and at least one X.

2. The character string must contain at least one 0 and at least one X.

3. The character string must contain at least one 0 (zero) and at least one A. Its contents, when represented in Standard Data Format, are allowable characters chosen from the EBCDIC set.

USAGE IS DISPLAY is used in conjunction with alphanumeric edited items.

If a VALUE clause is specified for an alphanumeric edited item, the literal is treated exactly as specified; no editing is performed.

Editing Rules: Alphanumeric edited items are subject to only one type of editing: simple insertion using the symbols 0 and B.

Examples of alphanumeric edited items:

PICTURE	Value of Data	Edited Result
000X(12)	ALPHANUMER01	000ALPHANUMER01
BBBX(12)	ALPHANUMER01	ALPHANUMER01
000A(12)	ALPHABETIC	000ALPHABETIC
X(5)BX(7)	ALPHANUMERIC	ALPHA NUMERIC

NUMERIC EDITED ITEMS: A numeric edited item is one whose PICTURE character string is restricted to certain combinations of the symbols:

    B   P   V   Z   0   9   ,   .   *   +   -   CR   DB $

The allowable combinations are determined from the order of precedence of symbols and editing rules.

The maximum number of digit positions that may be represented in the character string is 18.

The contents of the character positions that represent a digit, in Standard Data Format, must be one of the numerals.

USAGE IS DISPLAY is used in conjunction with numeric edited items.

If a VALUE clause is specified for a numeric edited item, the literal is treated exactly as specified; no editing is performed.

The maximum length of a numeric edited item is 127 characters.

Editing Rules: All types of editing are valid for numeric edited items.

Types of Editing

There are two general methods of performing editing in the PICTURE clause: by insertion or by suppression and replacement.

There are four types of _insertion_ editing:

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

Insertion Editing

Simple insertion editing is performed using the following insertion characters:

, (comma)   B (space)   0 (zero)

The insertion characters are counted in the size of the item and represent the position in the item into which the character will be inserted.

Examples of simple insertion editing:

PICTURE	Value of Data	Edited Result
99,999	12345	12,345
9,999,000	12345	2,345,000
99B999B000	1234	01 234 000
99B999B000	12345	12 345 000
99BBB999	123456	23 456

Special insertion editing is performed using the period (.) as the insertion character. The result of special insertion editing is the appearance of the insertion character in the item in the same position as shown in the character string.

In addition to being an insertion character, the period represents a decimal point for alignment purposes. The insertion character used for the actual decimal point is counted in the size of the item.

The use of both the assumed decimal point, represented by the symbol V, and the actual decimal point, represented by the period insertion character, in one PICTURE character string is not allowed.

Examples of special insertion editing:

PICTURE	Value of Data	Edited Result
999.99	1.234	001.23
999.99	12.34	012.34
999.99	123.45	123.45
999.99	1234.5	234.50

Fixed insertion editing is performed by using the following insertion characters:

    currency symbol $
    editing sign control symbols +   -   CR   DB

Only one currency symbol and only one of the editing sign control symbols can be used in a given PICTURE character string.

Fixed insertion editing results in the insertion character occupying the same character position in the edited item as it occupied in the PICTURE character string.

$ The currency symbol must be the leftmost character position to be counted in the size of the item, unless it is preceded by either a + or a - symbol.

+ or -   When either symbol is used, it must represent the leftmost or rightmost character position to be counted in the size of the item.

CR or DB   When either symbol is used, it represents two character positions in determining the size of the item and must represent the rightmost character positions that are counted in the size of the item.

Editing sign control symbols produce results depending upon the value of the data item as shown in Table 8.

Table 8.   Editing Sign Control Symbols and their Results

Editing Symbol in PICTURE Character String	Result	
	Data Item Positive or Zero	Data Item Negative
+	+	-
-	space	-
CR	2 spaces	CR
DB	2 spaces	DB

Examples of fixed insertion editing:

PICTURE	Value of Data	Edited Result
999.99+	+6555.556	555.55+
+9999.99	-5555.555	-5555.55
9999.99-	+1234.56	1234.56
$999.99	-123.45	$123.45
-$999.99	-123.456	-$123.45
$9999.99CR	+123.45	$0123.45
$9999.99DB	-123.45	$0123.45DB

Floating insertion editing is indicated in a PICTURE character string by using a string of at least two of the allowable insertion characters $ + or - to represent the leftmost numeric character positions into which the insertion characters can be floated.

The currency symbol ($) and the editing sign symbols (+ or -) are mutually exclusive as floating insertion characters in a given PICTURE character string.

Any of the simple insertion characters (, B 0) 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. Any or all leading numeric character positions to the left of the decimal point are represented by the insertion character.

2. All of the numeric character positions in the PICTURE character string are represented by the insertion character.

The result of floating insertion editing depends upon the representation in the PICTURE character string:

1. If the insertion characters are only to the left of the decimal point, a single insertion character is placed into the character position immediately preceding the first nonzero digit in the data represented by the insertion symbol string or the decimal point, whichever is farther to the left of the PICTURE character string.

2. If all numeric character positions in the PICTURE character string are represented by the insertion character, the result depends upon the value of the data. If the value is zero, the entire data item will contain spaces. If the value is not zero, the result is the same as when the insertion characters are only to the left of the decimal point.

To avoid truncation when using floating insertion editing, the programmer must specify the minimum size of the PICTURE character string for the receiving data item to be:

1. The number of characters in the sending item, plus

2. The number of insertion characters (other than floating insertion characters) being edited into the receiving data item, plus

3. One character for the floating insertion character.

Examples of floating insertion editing:

PICTURE	Value of Data	Edited Result
$$$$.99	.123	$.12
$$$9.99	.12	$0.12
$,$$$,999.99	-1234.56	$1,234.56
+,+++,999.99	-123456.789	-123,456.78
$$,$$$,$$$.99CR	-1234567	$1,234,567.00CR
$$,$$$,$$$.99DB	+1234567	$1,234,567.00
++,+++,+++.+++	0000.00	

Zero Suppression and Replacement Editing

Zero suppression and replacement editing means the suppression of leading zeros in numeric character positions and is indicated by the use of the alphabetic character Z or the character * in the PICTURE character string. If Z is used, the replacement character will be the space; if * is used, the replacement character will be *.

The symbols + - * Z and $ are mutually exclusive as floating replacement characters in a given PICTURE character string.

Each suppression symbol is counted in determining the size of an item.

Zero suppression and replacement editing is indicated in a PICTURE character string by using a string of one or more of either allowable symbol to represent leading numeric character positions, which are to be replaced when the associated character position in the data contains a zero. Any of the simple insertion characters embedded in the string of symbols or to the immediate right of this string are part of the string. Simple insertion or fixed insertion editing characters to the left of the string are not included.

In a PICTURE character string, there are only two ways of representing zero suppression:

1. Any or all of the leading numeric character positions to the left of the decimal point are represented by suppression symbols.

2. All of the numeric character positions in the PICTURE character string are represented by suppression symbols.

If the suppression symbols appear only to the left of the decimal point, any leading zero in the data which appears in a character position corresponding to a suppression symbol in the string is replaced by the replacement character. Suppression terminates at the first nonzero digit in the data or at the decimal point, whichever is encountered first.

If all numeric character positions in the PICTURE character string are represented by suppression symbols, 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 of the data is zero, the entire data item will be spaces if the suppression symbol is Z, or it will be asterisks (except for the actual decimal point) if the suppression symbol is *.

If the value of the data is zero and the asterisk is used as the suppression symbol, zero suppression editing overrides the function of the BLANK WHEN ZERO clause, if specified.

Examples of Zero Suppression and Replacement Editing:

PICTURE	Value of Data	Edited Result
ZZZZ.ZZ	0000.00	
****.**	0000.00	****.**
ZZZZ.99	0000.00	.00
****.99	0000.00	****.00
ZZ99.99	00000.00	00.00
Z,ZZZ.ZZ+	+123.456	123.45+
*,***.**+	-123.45	**123.45-
,*,***.**+	+12345678.9	*2,345,678.90+
$Z,ZZZ,ZZZ.ZZCR	+12345.67	$ 12,345.67
$B*,***,***.**BBDB	-12345.67	$ ***12,345.67 DB

Program Product Information -- Version 3

SIGN Clause

The SIGN clause specifies the position and mode of representation of the operational sign for a numeric data description entry.

```
r-----------------------------------------------------------------------1
|                              Format                                    |
|-----------------------------------------------------------------------|
|                                                                        |
|                          (  LEADING  )                                 |
|             [SIGN IS]  {            }  [SEPARATE CHARACTER]             |
|                          (  TRAILING )                                 |
L-----------------------------------------------------------------------J
```

The SIGN clause is required only when an explicit description of the
properties of the operational sign is necessary.

The numeric data description entries to which the SIGN clause
applies must, explicitly or implicitly, be described as USAGE IS
DISPLAY.

Only one SIGN clause may apply to any given numeric data description
entry.

The SIGN clause may be specified only for a numeric data description
entry whose PICTURE contains the character S, or for a group item
containing at least one such numeric data description entry.

When specified, the SIGN clause defines the position and mode of
representation of the operational sign for the numeric data
description entry to which it applies, or for each numeric data
description entry subordinate to the group to which it applies.

If the SEPARATE CHARACTER option is not specified, then:

*   The operational sign is presumed to be associated with the
    LEADING or TRAILING digit position, whichever is specified, of
    the elementary numeric data item. (In this instance,
    specification of SIGN IS TRAILING is the equivalent of the
    standard action of the compiler.)

*   The character S in the PICTURE character string is not counted
    in determining the size of the item (in terms of Standard Data
    Format characters).

If the SEPARATE CHARACTER option is specified, then:

*   The operational sign is presumed to be the LEADING or TRAILING
    character position, whichever is specified, of the elementary
    numeric data item. This character position is not a digit
    position.

*   The character S in the PICTURE character string is counted in
    determining the size of the data item (in terms of Standard Data
    Format characters).

*   + is the character used for the positive operational sign.

*   - is the character used for the negative operational sign.

*   At object time if one of the characters + or - is not present in
    the data an error occurs, and the program will terminate
    abnormally.

Every numeric data description entry whose PICTURE contains the
character S is a signed numeric data description entry. If the SIGN
clause applies to such an entry, and conversion is necessary for
purposes of computation, or for comparisons, conversion takes place
automatically.

If the SIGN clause is not specified for a numeric data description
entry whose PICTURE character string contains the character S then

the position of the operational sign is determined as explained in
the description of the USAGE clause.

SYNCHRONIZED Clause

The SYNCHRONIZED clause specifies the alignment of an elementary item
on one of the proper boundaries in core storage.

```
┌─────────────────────────────────────────────────────────────────────┐
│                              Format                                   │
├─────────────────────────────────────────────────────────────────────┤
│                                                                       │
│  ⎧ SYNCHRONIZED ⎫   ⎡ LEFT  ⎤                                          │
│  ⎨              ⎬   ⎢       ⎥                                          │
│  ⎩ SYNC         ⎭   ⎣ RIGHT ⎦                                          │
│                                                                       │
└─────────────────────────────────────────────────────────────────────┘
```
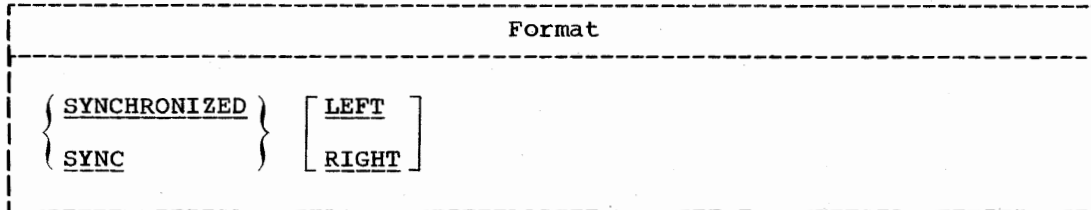
The SYNCHRONIZED clause is used to ensure efficiency when performing
arithmetic operations on an item.

The SYNCHRONIZED clause may appear only at the elementary level or at
level-01. When used at level-01, every elementary item within this
level-01 item is synchronized.

If either the LEFT or the RIGHT option is specified, it is treated as
comments.

The length of an elementary item is not affected by the SYNCHRONIZED
clause.

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

When the item is aligned, the character positions between the last
item assigned and the current item are known as "slack bytes." These
unused character positions are included in the size of any group to
which the elementary item preceding the synchronized elementary item
belongs.

The proper boundary used to align the item to be synchronized depends
on the format of the item as defined by the USAGE clause.

When the SYNCHRONIZED clause is specified, the following actions are
taken:

For a COMPUTATIONAL item:

1. If its PICTURE is in the range of S9 through S9(4), the item is
    aligned on a halfword (even) boundary.

2. If its PICTURE is in the range of S9(5) through S9(18), the item is
    aligned on a fullword (multiple of 4) boundary.

For a COMPUTATIONAL-1 item, the item is aligned on a fullword boundary.

For a COMPUTATIONAL-2 item, the item is aligned on a doubleword (multiple of 8) boundary.

For a DISPLAY or COMPUTATIONAL-3 item, the SYNCHRONIZED clause is treated as comments.

Note: In the discussion that follows, the term "computational" refers to COMPUTATIONAL, COMPUTATIONAL-1, and COMPUTATIONAL-2 items.

When the SYNCHRONIZED clause is specified for an item that also contains a REDEFINES clause, the data item that is redefined must have the proper boundary alignment for the data item that REDEFINES it. For example, if the programmer writes:

```
02  A              PICTURE X(4).
02  B   REDEFINES A PICTURE S9(9) COMP SYNC.
```

he must ensure that A begins on a fullword boundary.
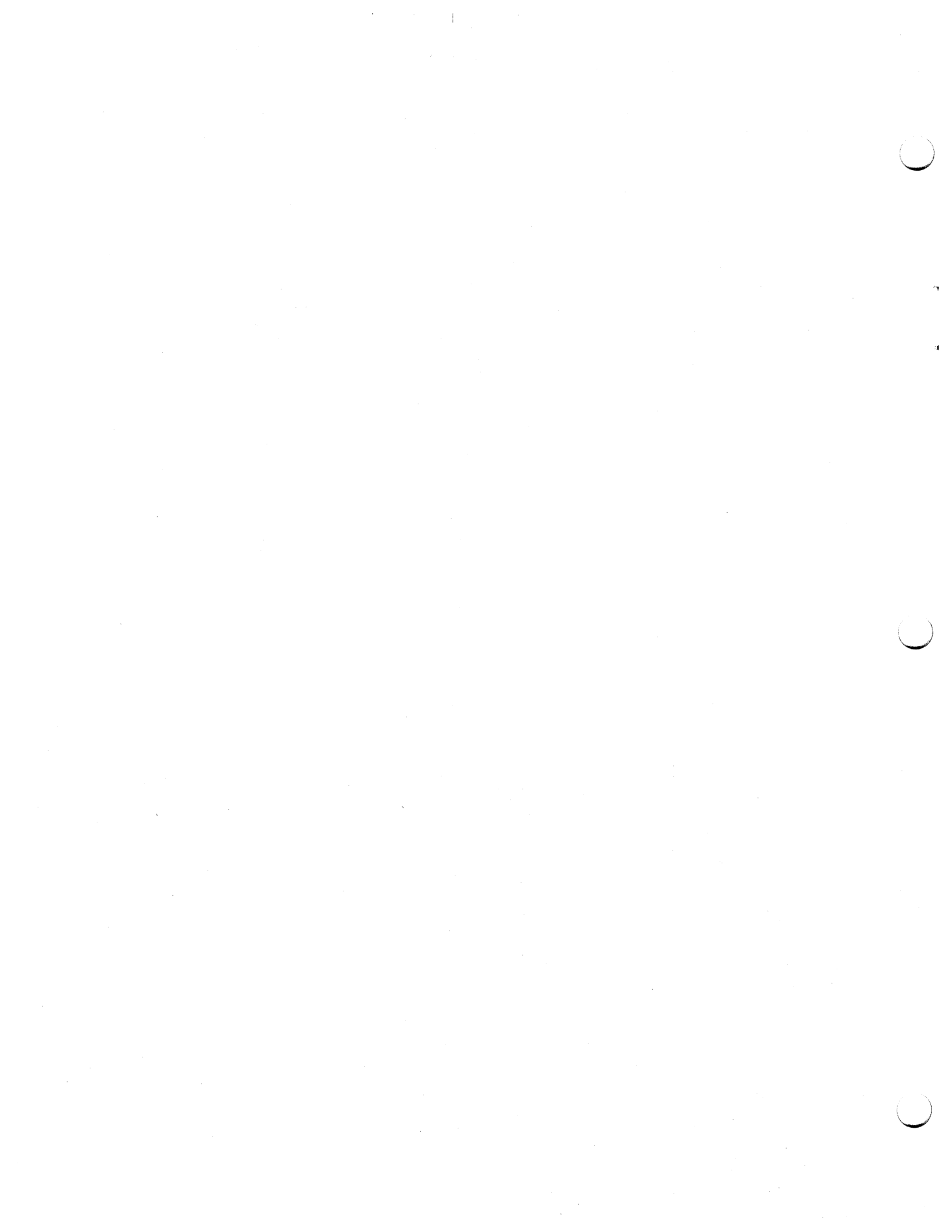
When the SYNCHRONIZED clause is specified for a computational item
that is the first elementary item subordinate to an item that contains a
REDEFINES clause, the computational item must not require the addition
of slack bytes.

When SYNCHRONIZED is not specified for binary or internal
floating-point items, no space is reserved for slack bytes. However,
when computation is done on these fields, the compiler generates the
necessary instructions to move the items to a work area which has the
correct boundary necessary for computation.

In the File Section, the compiler assumes that all level-01 records
containing SYNCHRONIZED items are aligned on a doubleword boundary in
the buffer. The user must provide the necessary inter-record slack
bytes to ensure alignment.

In the Working-Storage Section, the compiler will align all level-01
entries on a doubleword boundary.

For the purposes of aligning COMPUTATIONAL, COMPUTATIONAL-1, and
COMPUTATIONAL-2 items in the Linkage Section, all level-01 items are
assumed to begin on doubleword boundaries. Therefore, if the user
issues a CALL statement he must ensure that such operands of any USING
clause within it are correspondingly aligned.

Slack Bytes

There are two types of slack bytes: intra-record slack bytes and
inter-record slack bytes.

Intra-record slack bytes are unused character positions preceding
each synchronized item in the record.

Inter-record slack bytes are unused character positions added between
blocked logical records.

INTRA-RECORD SLACK BYTES: For an output file, or in the Working-Storage
Section, the compiler inserts intra-record slack bytes to ensure that
all SYNCHRONIZED items are on their proper boundaries. For an input
file, or in the Linkage Section, the compiler expects intra-record slack
bytes to be present when necessary to assure the proper alignment of a
SYNCHRONIZED item.

Because it is important for the user to know the length of the
records in a file, the algorithm the compiler uses to determine whether
slack bytes are required and, if they are required, the number of slack
bytes to add, is as follows:

- The total number of bytes occupied by all elementary data items
  preceding the computational item are added together, including
  any slack bytes previously added.

- This sum is divided by m, where:

  $m = 2$ for COMPUTATIONAL items of four-digit length or less

  $m = 4$ for COMPUTATIONAL items of five-digit length or more

  $m = 4$ for COMPUTATIONAL-1 items

  $m = 8$ for COMPUTATIONAL-2 items

- If the remainder (r) of this division is equal to zero, no slack bytes are required. If the remainder is not equal to zero, the number of slack bytes that must be added is equal to $m - r$.

These slack bytes are added to each record immediately following the elementary data item preceding the computational item. They are defined as if they were an item with a level number equal to that of the elementary item that immediately precedes the SYNCHRONIZED item, and are included in the size of the group which contains them.

For example:

```
01   FIELD-A.
     02   FIELD-B            PICTURE X(5).
     02   FIELD-C.
          03   FIELD-D       PICTURE XX.
          [03   Slack-Bytes  PICTURE X.   Inserted by compiler]
          03   FIELD-E       PICTURE S9(6) COMP SYNC.

01   FIELD-L.
     02   FIELD-M            PICTURE X(5).
     02   FIELD-N            PICTURE XX.
     [02   Slack-Bytes       PICTURE X.   Inserted by compiler]
     02   FIELD-O.
          03   FIELD-P       PICTURE S9(6) COMP SYNC.
```

Slack bytes may also be added by the compiler when a group item is defined with an OCCURS clause and contains within it a SYNCHRONIZED data item with USAGE defined as COMPUTATIONAL, COMPUTATIONAL-1, or COMPUTATIONAL-2. To determine whether slack bytes are to be added, the following action is taken:

- The compiler calculates the size of the group, including all the necessary intra-record slack bytes.

- This sum is divided by the largest m required by any elementary item within the group.

- If r is equal to zero, no slack bytes are required. If r is not equal to zero, $m - r$ slack bytes must be added.

The slack bytes are inserted at the end of each occurrence of the group item containing the OCCURS clause. For example, if a record is defined as follows:

```
01   WORK-RECORD.
     02   WORK-CODE               PICTURE X.
     02   COMP-TABLE OCCURS 10 TIMES.
          03   COMP-TYPE          PICTURE X.
          [03   Ia-Slack-Bytes    PICTURE XX.   Inserted by compiler]
          03   COMP-PAY           PICTURE S9(4)V99 COMP SYNC.
          03   COMP-HRS           PICTURE S9(3) COMP SYNC.
          03   COMP-NAME          PICTURE X(5).
```

The record will appear in storage as shown in Figure 8
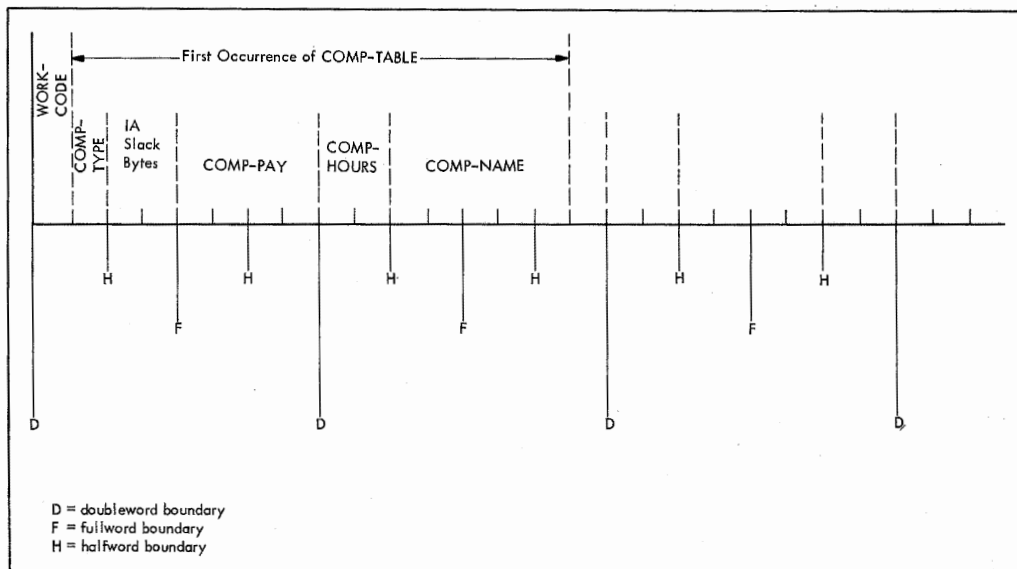
Figure 8. Insertion of Intra-occurrence Slack Bytes

In order to align COMP-PAY and COMP-HRS upon their proper boundaries, the compiler has added two intra-occurrence slack bytes (shown above as Ia-Slack-Bytes).

However, without further adjustment, the second occurrence of COMP-TABLE would now begin one byte before a doubleword boundary, and the alignment of COMP-PAY and COMP-HRS would not be valid for any occurrence of the table after the first. Therefore, the compiler must add inter-occurrence slack bytes at the end of the group, as though the record had been written:

```
01  WORK-RECORD.
    02  WORK-CODE             PICTURE X.
    02  COMP-TABLE OCCURS 10  TIMES.
        03  COMP-TYPE         PICTURE X.
        [03 Ia-Slack-Bytes    PICTURE XX.   Inserted by compiler]
        03  COMP-PAY          PICTURE S9(4)V99 COMP SYNC.
        03  COMP-HRS          PICTURE S9(3)  COMP SYNC.
        03  COMP-NAME         PICTURE X(5).
        [03 Ie-Slack-Bytes    PICTURE XX.   Inserted by compiler]
```

so that the second (and each succeeding) occurrence of COMP-TABLE begins one byte beyond a doubleword boundary. The storage layout for the first occurrences of COMP-TABLE will now appear as shown in Figure 9.
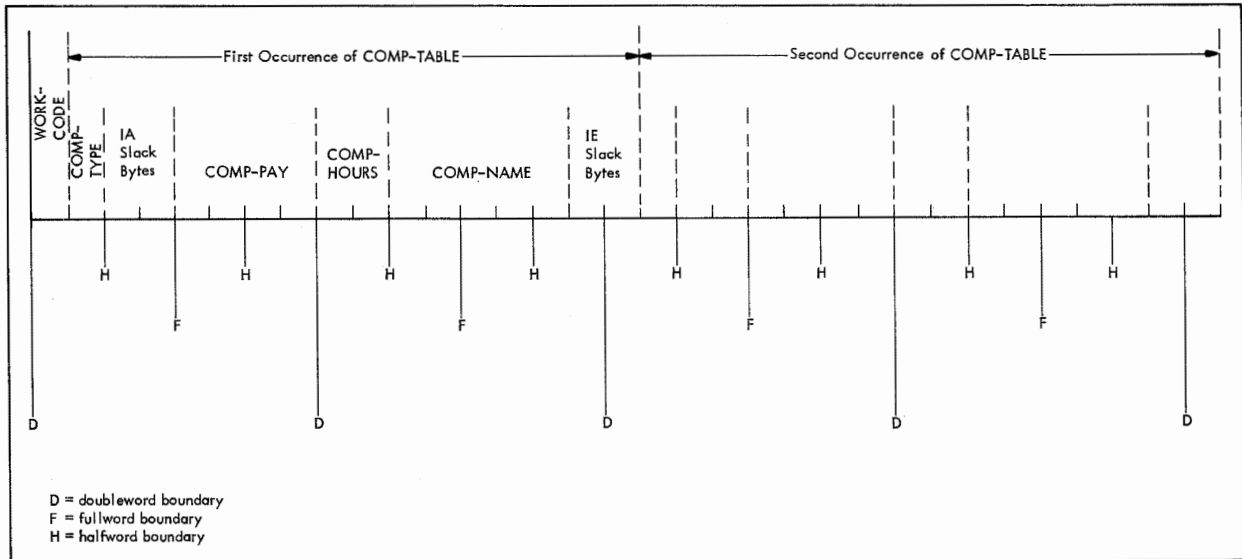
Figure 9. Insertion of Inter-occurrence Slack Bytes

Each succeeding occurrence within the table will now begin at the same relative position to word boundaries as the first.

Where SYNCHRONIZED data items defined as COMPUTATIONAL, COMPUTATIONAL-1, or COMPUTATIONAL-2 follow an entry containing an OCCURS clause with the DEPENDING ON option, slack bytes are added on the basis of the field occurring the maximum number of times. If the length of this field is not divisible by the $\underline{m}$ required for the computational data, only certain values of the data-name that is the object of the DEPENDING ON option will give proper alignment of the computational fields. These values are those for which the length of the field times the number of occurrences plus the slack bytes that have been calculated based on the maximum number of occurrences is divisible by $\underline{m}$.

For example:

```
01   FIELD-A.
     02   FIELD-B                PICTURE 99.
     02   FIELD-C                PICTURE X OCCURS 20 TO 99 TIMES
          DEPENDING ON FIELD-B.
     [02  Slack-Byte             PICTURE X.   Inserted by compiler]
     02   FIELD-D                PICTURE S99 COMP SYNC.
```

In this example, when references to FIELD-D are required, FIELD-B is restricted to odd values only.

```
01   FIELD-A.
     02   FIELD-B                PICTURE 999.
     02   FIELD-C                PICTURE XX OCCURS 20 TO 99 TIMES
          DEPENDING ON FIELD-B.
     [02  Slack-Byte             PICTURE X.   Inserted by compiler]
     02   FIELD-D                PICTURE S99 COMP SYNC.
```

In this example all values of FIELD-B give proper references to FIELD-D.

INTER-RECORD SLACK BYTES: If the file contains blocked logical records that are to be processed in a buffer, and any of the records contain entries defined as COMPUTATIONAL, COMPUTATIONAL-1, or COMPUTATIONAL-2, the user must add any inter-record slack bytes needed for proper alignment.

The lengths of all the elementary data items in the record, including all intra-record slack bytes, are added. For mode V records, it is necessary to add four bytes for the count field. The total is then divided by the highest value of m for any one of the elementary items in the record.

If r (the remainder) is equal to zero, no inter-record slack bytes are required. If r is not equal to zero, $m - r$ slack bytes are required. These slack bytes may be specified by writing a level-02 FILLER at the end of the record.

Example: The following example shows the method of calculating both intra-record and inter-record slack bytes. Consider the following record description:

```
01   COMP-RECORD.
     02   A-1      PICTURE X(5).
     02   A-2      PICTURE X(3).
     02   A-3      PICTURE X(3).
     02   B-1      PICTURE S9999   USAGE COMP SYNCHRONIZED.
     02   B-2      PICTURE S99999   USAGE COMP SYNCHRONIZED.
     02   B-3      PICTURE S9999   USAGE COMP SYNCHRONIZED.
```

The number of bytes in A-1, A-2, and A-3 total 11. B-1 is a 4-digit COMPUTATIONAL item and, therefore, one intra-record slack byte must be added before B-1. With this byte added, the number of bytes preceding B-2 total 14. Since B-2 is a COMPUTATIONAL item of 5 digits in length, two intra-record slack bytes must be added before it. No slack bytes are needed before B-3.

The revised record description entry now appears as:

```
01   COMP-RECORD.
     02   A-1           PICTURE X(5).
     02   A-2           PICTURE X(3).
     02   A-3           PICTURE X(3).
    [02   Slack-Byte-1  PICTURE X.   Inserted by compiler]
     02   B-1           PICTURE S9999   USAGE COMP SYNCHRONIZED.
    [02   Slack-Byte-2  PICTURE XX.   Inserted by compiler]
     02   B-2           PICTURE S99999 USAGE COMP SYNCHRONIZED.
     02   B-3           PICTURE S9999   USAGE COMP SYNCHRONIZED.
```

There are a total of 22 bytes in COMP-RECORD, but from the rules given in the preceding discussion, it appears that $m = 4$ and $r = 2$. Therefore, to attain proper alignment for blocked records, the user must add two inter-record slack bytes at the end of the record.

The final record description entry appears as:

```
01   COMP-RECORD.
     02   A-1           PICTURE X(5).
     02   A-2           PICTURE X(3).
     02   A-3           PICTURE X(3).
    [02   Slack-Byte-1  PICTURE X.   Inserted by compiler]
     02   B-1           PICTURE S9999   USAGE COMP SYNCHRONIZED.
    [02   Slack-Byte-2  PICTURE XX.   Inserted by compiler]
     02   B-2           PICTURE S99999 USAGE COMP SYNCHRONIZED.
     02   B-3           PICTURE S9999   USAGE COMP SYNCHRONIZED.
     02   FILLER        PICTURE XX. [inter-record slack bytes added by
                                     user]
```

USAGE Clause

The USAGE clause specifies the manner in which a data item is represented in core storage.

```
+-----------------------------------------------------------------------+
|                               Format 1                                |
+-----------------------------------------------------------------------+
|                                                                       |
|                        DISPLAY                                        |
|                       (COMPUTATIONAL)                                 |
|                       (COMP        )                                  |
|                       (COMPUTATIONAL-1)                               |
|                       (COMP-1       )                                 |
|   [USAGE IS]          (COMPUTATIONAL-2)                               |
|                       (COMP-2       )                                 |
|                       (COMPUTATIONAL-3)                               |
|                       (COMP-3       )                                 |
|                        DISPLAY-ST                                     |
|                        INDEX                                          |
|                                                                       |
+-----------------------------------------------------------------------+
```

```
+-----------------------------------------------------------------------+
|                        Format 2 (Version 3)                           |
+-----------------------------------------------------------------------+
|                                                                       |
|                       (COMPUTATIONAL-4)                               |
|   [USAGE IS]          (              )                                |
|                       (COMP-4        )                                |
|                                                                       |
+-----------------------------------------------------------------------+
```
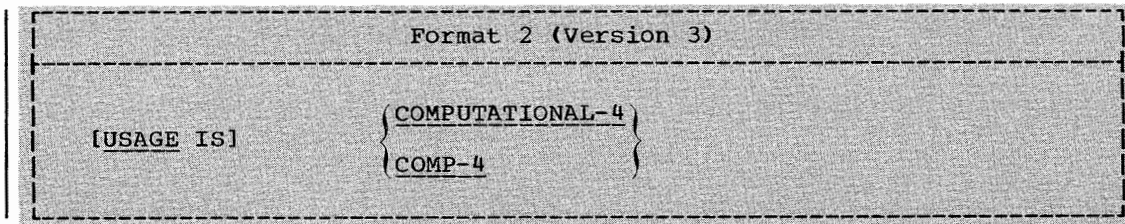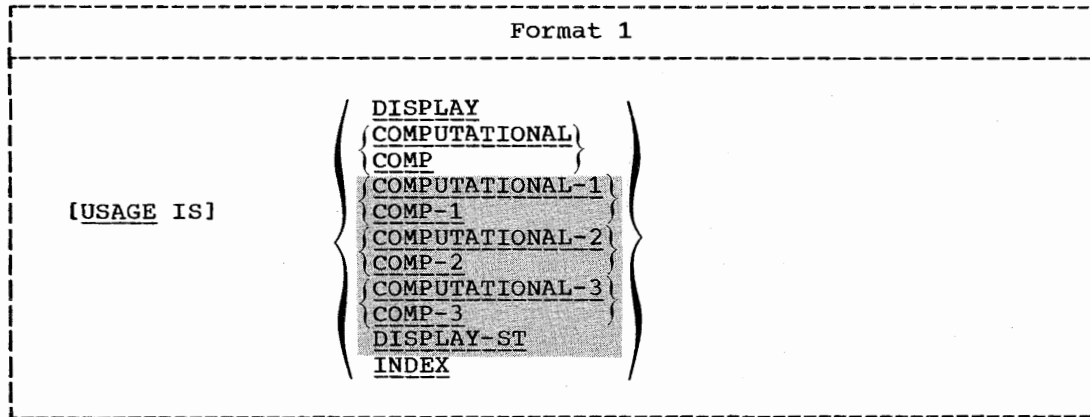
The USAGE clause can be specified at any level of data description. However, if the USAGE clause is written at a group level, it applies to each elementary item in the group. The usage of an elementary item cannot contradict the usage of a group to which an elementary item belongs.

This clause specifies the manner in which a data item is represented in core storage. However, the specifications for some statements in the Procedure Division may restrict the USAGE clause of the operand referred to.

If the USAGE clause is not specified for an elementary item, or for any group to which the item belongs, it is assumed that the usage is DISPLAY.

Note: ASCII considerations for the USAGE clause are given in Appendix E.

DISPLAY OPTION

The DISPLAY option can be explicit or implicit. It specifies that the data item is stored in character form, one character per eight-bit byte. This corresponds to the form in which information is represented for initial card input or for final printed or punched output. USAGE IS DISPLAY is valid for the following types of items:

- alphabetic

- alphanumeric

- alphanumeric edited

- numeric edited

- external decimal

- external floating point

   The alphabetic, alphanumeric, alphanumeric edited, and numeric edited items are discussed in the description of the PICTURE clause.

External Decimal Items: These items are sometimes referred to as zoned decimal items. Each digit of a number is represented by a single byte. The four high-order bits of each byte are zone bits; the four high-order bits of the low-order byte represent the sign of the item. The four low-order bits of each byte contain the value of the digit. When external decimal items are used for computations, the compiler performs the necessary conversions.

   The maximum length of an external decimal item is 18 digits.

   Examples of external decimal items and their internal representation are shown in Table 8.1.

External Floating-Point Items: The PICTURE of an external floating-point item is in the following form:

     {±}mantissa E{±}exponent

(See the discussion of the PICTURE clause for the valid combination of symbols.)

   The mantissa is the decimal part of the number.

   The exponent specifies a power of ten that is used as a multiplier.

   The value of an external floating-point number is the mantissa multiplied by the power of ten expressed by the exponent. The magnitude of a number represented by a floating-point item must be greater than $5.4 \times (10^{-79})$ but must not exceed $.72 \times (10^{76})$.

   When used as a numeric operand an external floating-point number is scanned at object time, and converted to the equivalent internal floating-point values. In this form, the number is used in arithmetic operations. (See COMPUTATIONAL-1 and COMPUTATIONAL-2 options.)

   An example of an external floating-point number and its internal representation is shown in Table 8.1.

The Computational Options

   A COMPUTATIONAL, COMPUTATIONAL-1, COMPUTATIONAL-2, COMPUTATIONAL-3 or COMPUTATIONAL-4 item represents a value to be used in arithmetic operations and must be numeric. If the USAGE of any group item is described with any of these options, it is the elementary items within this group which have that USAGE. The group item itself cannot be used in computations.

COMPUTATIONAL OPTION: This option is specified for binary data items. Such items have a decimal equivalent consisting of the decimal digits 0 through 9, plus a sign.

The amount of storage occupied by a binary item depends on the number of decimal digits defined in its PICTURE clause:

Digits in PICTURE Clause	Storage Occupied
1 through 4	2 bytes (halfword)
5 through 9	4 bytes (fullword)
10 through 18	8 bytes (2 fullwords -- not necessarily a doubleword)

The leftmost bit of the storage area is the operational sign.

The PICTURE of a COMPUTATIONAL item may contain only 9's, the operational sign character S, the implied decimal point V, and one or more P's.

An example of a binary item is shown in Table 8.1.

Note: The COMPUTATIONAL option is system dependent; for this compiler it is binary.

COMPUTATIONAL-1, COMPUTATIONAL-2 OPTIONS: These options are specified for internal floating-point items. Such an item is equivalent to an external floating-point item in capability and purpose. Such items occupy either 4 or 8 bytes of storage.

The sign of the fraction (mantissa) is the leftmost bit in either format.

The exponent appears in bit positions 1 through 7.

The fraction -- equivalent to the mantissa -- appears in the rightmost bytes.

COMPUTATIONAL-1 is specified for short-precision internal floating point items. Such items are four bytes in length, aligned on a fullword boundary. The fraction occupies the rightmost three bytes of the item.

COMPUTATIONAL-2 is specified for long-precision floating-point items. Such items are eight bytes in length, and are aligned on a doubleword boundary. The fraction occupies the rightmost seven bytes of the item.

No PICTURE cluase may be associated with an internal floating-point item.

Examples of internal floating-point items, and their internal representation, are shown in Table 8.1.

COMPUTATIONAL-3 OPTION: This option is specified for internal decimal items. Such an item appears in storage in packed decimal format. There are two digits per byte, with the sign contained in the low-order four bits of the rightmost byte. Such an item may contain any of the digits 0 through 9, plus a sign, representing a value not exceeding 18 decimal digits.

For internal decimal items whose PICTURE does not contain an S, the sign position is occupied by a bit configuration that is interpreted as positive, but that does not represent an overpunch.

The PICTURE of a COMPUTATIONAL-3 item may contain only 9's, the operational sign character S, the assumed decimal point V, and one or more P's.

Examples of internal-decimal items and their internal representation are shown in Table 8.1.

Program Product Information -- Version 3

COMPUTATIONAL-4 OPTION: This option (Format 2) is specified for system-independent binary items. For this compiler, it is the equivalent of COMPUTATIONAL.

USAGE DISPLAY-ST is discussed in the chapter on Sterling Currency.
USAGE INDEX is discussed in the chapter on Table Handling.

•Table 8.1.   Internal Representation of Numeric Items (Part 1 of 2)

Item	Value	Description	Internal Representation*
External Decimal	-1234	DISPLAY PICTURE 9999	|Z1|Z2|Z3|F4| byte
		DISPLAY PICTURE S9999	|Z1|Z2|Z3|D4| byte Note that, internally, the D4, which represents -4, is the same bit configuration as the EBCDIC character M.
		(Version 3) DISPLAY PICTURE S9999 SIGN LEADING	|D1|Z2|Z3|Z4| byte Note that internally the D1, which represents -1, is the same bit configuration as the EBCDIC character J.
		(Version 3) DISPLAY PICTURE S9999 SIGN TRAILING SEPARATE	|Z1|Z2|Z3|Z4|-| byte
Binary	-1234	COMPUTATIONAL PICTURE S9999	|1111|1011|0010|1110| S byte
		(Version 3) COMP-4 PICTURE S9999	Note that, internally, negative binary numbers appear in two's complement form.

*Codes used in this column are as follows:
    Z = zone, equivalent to hexadecimal F, bit configuration 1111

    Hexadecimal numbers and their equivalent meanings are:
        F = nonprinting plus sign (treated as an absolute value)
        C = internal equivalent of plus sign, bit configuration 1100
        D = internal equivalent of minus sign, bit configuration 1101

    S = sign position of a numeric field; internally,
            1 in this position means the number is negative
            0 in this position means the number is positive

    b = a blank

Table 8.1. Internal Representation of Numeric Items (Part 2 of 2)

Item	Value	Description	Internal Representation*
Internal Decimal	+1234	COMPUTATIONAL-3 PICTURE 9999	|01|23|4F| byte
		COMPUTATIONAL-3 PICTURE S9999	|01|23|4C| byte
External Floating-Point	+12.34E+2	DISPLAY PICTURE +99.99E-99	|+|1|2|.|3|4|E|b|0|2| byte
Internal Floating-Point		COMPUTATIONAL-1	|S|Exponent|Fraction| 0 1 7 8 31
		COMPUTATIONAL-2	|S|Exponent|Fraction| 0 1 7 8 63

*Codes used in this column are as follows:
    Z = zone, equivalent to hexadecimal F, bit configuration 1111

    Hexadecimal numbers and their equivalent meanings are:
        F = nonprinting plus sign (treated as an absolute value)
        C = internal equivalent of plus sign, bit configuration 1100
        D = internal equivalent of minus sign, bit configuration 1101

    S = sign position of a numeric field; internally,
            1 in this position means the number is negative
            0 in this position means the number is positive

    b = a blank

VALUE Clause

    The VALUE clause is used to define the initial value of a
Working-Storage item or the value associated with a condition-name.

    There are two formats of the VALUE clause:

```
+-----------------------------------------------------------------------+
|                             Format 1                                  |
+-----------------------------------------------------------------------+
|                                                                       |
|    VALUE IS literal                                                   |
|                                                                       |
+-----------------------------------------------------------------------+
```

```
+-----------------------------------------------------------------------+
|                             Format 2                                  |
+-----------------------------------------------------------------------+
|                                                                       |
|  ( VALUE IS   )                                                       |
|  {           }      literal-1 [THRU literal-2]                        |
|  ( VALUES ARE )                                                       |
|                                                                       |
|         [literal-3 [THRU literal-4]]...                               |
|                                                                       |
+-----------------------------------------------------------------------+
```

    The VALUE clause must not be stated for any item whose size, explicit
or implicit, is variable.

    A figurative constant may be substituted wherever a literal is
specified.

    Rules governing the use of the VALUE clause differ with the
particular section of the Data Division in which it is specified.

    1. In the File Section and the Linkage Section, the VALUE clause must
        be used only in condition-name entries. However, this compiler
        will accept the VALUE clause in other File Section and Linkage
        Section entries, and treat it as comments.

    2. In the Working-Storage Section, the VALUE clause must be used in
        condition-name entries, and it may also be used to specify the
        initial value of any data item. It causes the item to assume the
        specified value at the start of execution of the object program.
        If the VALUE clause is not used in an item's description, the
        initial value is unpredictable.

    3. In the Report Section, the VALUE clause causes the report data item
        to assume the specified value each time its report group is
        presented. This clause may be used only at an elementary level in
        the Report Section. The Report Section is discussed in detail in
        the "Report Writer" chapter.

    The VALUE clause must not be specified in a data description entry
that contains an OCCURS clause or in an entry that is subordinate to an
entry containing an OCCURS clause. This rule does not apply to
condition-name entries.

    Within a given record description, the VALUE clause must not be used
in a data description entry that is subsequent to a data description
entry which contains an OCCURS clause with a DEPENDING ON phrase.

The VALUE clause must not be specified in a data description entry which contains a REDEFINES clause or in an entry which is subordinate to an entry containing a REDEFINES clause. This rule does not apply to condition-name entries.

If the VALUE clause is used in an entry at the group level, the literal must be a figurative constant or a nonnumeric literal, and the group area is initialized without consideration for the USAGE of the individual elementary or group items contained within this group. The VALUE clause then cannot be specified at subordinate levels within this group.

The VALUE clause cannot be specified for a group containing items with descriptions including JUSTIFIED, SYNCHRONIZED, or USAGE (other than USAGE IS DISPLAY).

The VALUE clause must not be specified for external floating-point items.

The following rules apply:

1. If the item is numeric, all literals in the VALUE clause must be numeric literals. If the literal defines the value of a Working-Storage item, the literal is aligned according to the rules for numeric moves, except that the literal must not have a value that would require truncation of nonzero digits.

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

3. All numeric literals in a VALUE clause of an item must have a value that is within the range of values indicated by the PICTURE clause for that item. For example, for PICTURE 99PPP, the literal must be within the range 1000 through 99000 or zero. For PICTURE PPP99, the literal must be within the range .00000 through .00099.

4. The function of the editing characters in a PICTURE clause is ignored in determining the initial appearance of the item described. However, editing characters are included in determining the size of the item.

Table 12. Permissible Comparisons

First Operand \ Second Operand	GR	AL	AN	ANE	NE	FC* NNL	ZR NL	ED	BI	ID	EF	IF	SR	SN	IN	IDI
Group (GR)	NN	NN	NN	NN	NN	NN	NN	NN	NN	NN	NN	NN	NN	NN		
Alphabetic (AL)	NN	NN	NN	NN	NN	NN	NN	NN			NN		NN	NN		
Alphanumeric (AN)	NN	NN	NN	NN	NN	NN	NN	NN			NN		NN	NN		
Alphanumeric Edited (ANE)	NN	NN	NN	NN	NN	NN	NN	NN			NN		NN	NN		
Numeric Edited (NE)	NN	NN	NN	NN	NN	NN	NN	NN			NN		NN	NN		
Figurative Constant (FC)* & Nonnumeric Literal (NNL)	NN	NN	NN	NN	NN			NN			NN		NN	NN		
Fig. Constant ZERO (ZR) & Numeric Literal (NL)	NN	NN	NN	NN	NN			NU	NU	NU	NU	NU	NN	NU	IO[1]	
External Decimal (ED)	NN	NN	NN	NN	NN	NN	NU	NU	NU	NU	NU	NU	NN	NU	IO[1]	
Binary (BI)	NN						NU	NU	NU	NU	NU	NU		NU	IO[1]	
Internal Decimal (ID)	NN						NU	NU	NU	NU	NU	NU		NU	IO[1]	
External Floating Point (EF)	NN	NN	NN	NN	NN	NN	NU	NU	NU	NU	NU	NU	NN	NU		
Internal Floating Point (IF)	NN						NU	NU	NU	NU	NU	NU		NU		
Sterling Report (SR)	NN	NN	NN	NN	NN	NN	NN	NN			NN		NN	NN		
Sterling Nonreport (SN)	NN	NN	NN	NN	NN	NN	NU	NU	NU	NU	NU	NU	NN	NU		
Index Name (IN)							IO[1]	IO[1]	IO[1]	IO[1]					IO	IV
Index Data Item (IDI)															IV	IV

*FC includes all Figurative Constants except ZERO.
[1]Valid only if the numeric item is an integer.

   NN = comparison as described for nonnumeric operands
   NU = comparison as described for numeric operands
   IO = comparison as described for two index-names
   IV = comparison as described for index data items

Sign Condition

The sign condition determines whether or not the algebraic value of a numeric operand (i.e., an item described as numeric) is less than, greater than, or equal to zero.

```
+----------------------------------------------------------------------+
|                               Format                                 |
+----------------------------------------------------------------------+
|                                                                      |
|  (identifier            )              (POSITIVE)                     |
|  {                      }   IS [NOT]   {NEGATIVE}                     |
|  (arithmetic-expression )              (ZERO    )                     |
|                                                                      |
+----------------------------------------------------------------------+
```

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. An unsigned field is positive or zero.

Switch-Status Condition

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

```
+----------------------------------------------------------------------+
|                               Format                                 |
+----------------------------------------------------------------------+
|                                                                      |
|    condition-name                                                    |
|                                                                      |
+----------------------------------------------------------------------+
```

The SPECIAL-NAMES paragraph of the Environment Division associates an ON or OFF value (condition-name) with a switch (function-name). The switch-status condition tests the value associated with the switch. The result of the test is true if the switch is set to the position corresponding to condition-name.

COMPOUND CONDITIONS

Two or more simple conditions can be combined to form a compound condition. Each simple condition is separated from the next by one of the logical operators AND or OR.

The logical operators must be preceded by a space and followed by a space. The meaning of the logical operators is as follows:

Logical Operator	Meaning
OR	logical inclusive OR, i.e., either or both are true
AND	logical conjunction, i.e., both are true
NOT	logical negation

Figure 10 shows the relationships between the logical operators and simple conditions A and B, where A and B have the following values:

Values for A	Values for B
True	True
False	True
True	False
False	False

A AND B	A OR B	NOT A	NOT (A AND B)	NOT A AND B	NOT (A OR B)	NOT A OR B
True	True	False	False	False	False	True
False	True	True	True	True	False	True
False	True	False	True	False	False	False
False	False	True	True	False	True	True

Figure 10. Logical Operators and the Resulting Values upon Evaluation

EVALUATION RULES

Logical evaluation begins with the least inclusive pair of parentheses and proceeds to the most inclusive.

If the order of evaluation is not specified by parentheses, the expression is evaluated in the following order:

1. Arithmetic expressions

2. Relational-operators

3. [NOT] condition

4. AND and its surrounding conditions are evaluated first, starting at the left of the expression and proceeding to the right.

5. OR and its surrounding conditions are then evaluated, also proceeding from left to right.

Consider the expression:

A IS NOT GREATER THAN B OR A + B IS EQUAL TO C AND D IS POSITIVE

This will be evaluated as if it were parenthesized as follows:

(A IS NOT GREATER THAN B) OR (((A + B) IS EQUAL TO C) AND (D IS POSITIVE)).

Compound Conditions

The order of evaluation is as follows:

1. (A + B) is evaluated, giving some intermediate result, for example, x.

2. (A IS NOT GREATER THAN B) is evaluated, giving some intermediate truth value, for example, t1.

3. (x IS EQUAL TO C) is evaluated, giving some intermediate truth value, for example, t2.

4. (D IS POSITIVE) is evaluated, giving some intermediate truth value, for example, t3.

5. (t2 AND t3) is evaluated, giving some intermediate truth value, for example, t4.

6. (t1 OR t4) is evaluated, giving the final truth value, and the result of the expression.

Table 13 shows permissible symbol pairs. A symbol pair in a compound condition is the occurrence of two symbols appearing in sequence.

Table 13. Permissible Symbol Pairs

First Symbol \ Second Symbol	Condition	OR	AND	NOT	(	)
Condition	-	p	p	-	-	p
OR	p	-	-	p	p	-
AND	p	-	-	p	p	-
NOT	p	-	-	-	p	-
(	p	-	-	p	p	-
)	-	p	p	-	-	p

p indicates a permissible pairing
- indicates that the pairing is not permitted

IMPLIED SUBJECTS AND RELATIONAL-OPERATORS

When relation conditions are written in a consecutive sequence, any relation condition except the first may be abbreviated by:

1. The omission of the subject of the relation condition, or

2. The omission of the subject and relational-operator of the relation condition.

Within a sequence of relation conditions, both forms of abbreviation may be used. The effect of using such abbreviations is as if the omitted subject was taken from the most recently stated subject, or the omitted relational-operator was taken from the most recently stated relational-operator.

```
┌─────────────────────────────────────────────────────────────────────────┐
│Format of Implied Subject:                                                 │
├─────────────────────────────────────────────────────────────────────────┤
│                                                                           │
│ ...subject   relational-operator    object                                │
│                                                                           │
│   ( AND )                                                                 │
│   {     } [NOT] relational-operator    object...                          │
│   ( OR  )                                                                 │
│                                                                           │
└─────────────────────────────────────────────────────────────────────────┘
```

```
┌─────────────────────────────────────────────────────────────────────────┐
│Format of Implied Subject and Relational-operator:                         │
├─────────────────────────────────────────────────────────────────────────┤
│                                                                           │
│                                              ( AND )                       │
│ ...subject      relational-operator  object  {     }  [NOT] object...      │
│                                              ( OR  )                       │
│                                                                           │
└─────────────────────────────────────────────────────────────────────────┘
```

Ambiguity may result from using NOT in conjunction with abbreviations. In this event, NOT is interpreted as a logical operator rather than as part of a relational-operator. For example, A > B AND NOT > C OR D is equivalent to either A > B AND NOT A > C OR A > D, or A > B AND (NOT A > C) OR A > D.

The following are examples of implied subjects, and relational-operators. Each example consists of two equivalent statements:

Implied Subject

```
A = B OR NOT > C            (The subject, A, is implied.)
A = B OR NOT A > C          (The subject, A, is explicit.)
```

Implied Subject and Relational Operator

```
A = B AND C                 (Subject and relational-operator, A = , are
                             implied.)
A = B AND A = C             (Subject and relational-operator, A = , are
                             explicit.)
```

Implied Subject, and Subject and Relational-Operator

```
A > B AND NOT < C AND D     (Subject, A, is implied in the second condi-
                             tion.  Subject, A, and relational-operator,
                             <, are implied in the third condition.)

A > B AND NOT A < C         (Subject, A, and relational-operator, <, are
   AND A < D                 explicit.)
```

The omitted subject is taken from the most recently stated subject, i.e., A.

The omitted relational-operator is taken from the most recently stated relational-operator, i.e., <.

CONDITIONAL STATEMENTS

A conditional statement specifies that the truth value of a condition is to be determined and that the subsequent action of the object program is dependent on this truth value. Conditional statements are listed in "Categories of Statements."

A conditional sentence is a conditional statement optionally preceded by an imperative statement, terminated by a period followed by a space.

Only the IF statement is discussed in this section. Discussion of the other conditional statements is included as part of the description of the associated imperative statements.

IF Statement

The IF statement causes a condition to be evaluated. The subsequent action of the object program depends upon whether the condition is true or false.
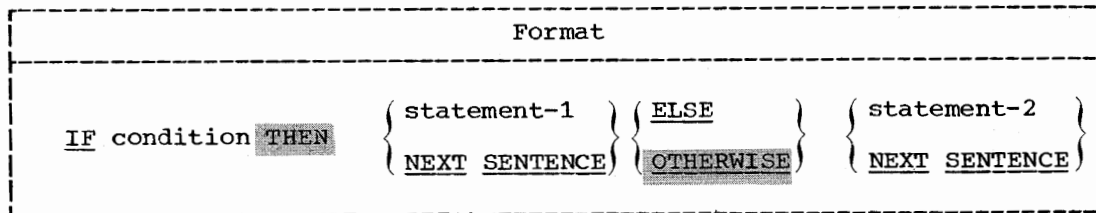
```
+---------------------------------------------------------------------+
|                               Format                                |
+---------------------------------------------------------------------+
|                                                                     |
|              ( statement-1  ) ( ELSE      ) ( statement-2  )         |
|  IF condition THEN {            }{          }{              }        |
|              ( NEXT SENTENCE ) ( OTHERWISE ) ( NEXT SENTENCE )       |
|                                                                     |
+---------------------------------------------------------------------+
```

The phrase ELSE/OTHERWISE NEXT SENTENCE may be omitted if it immediately precedes the period for the sentence.

When an IF statement is executed, the following action is taken:

1. If the condition is true, the statement immediately following the condition or THEN (statement-1) is executed. Control is then passed implicitly to the next sentence unless GO TO procedure-name is specified in statement-1. If the condition is true and NEXT SENTENCE is written, control passes explicitly to the next sentence.

2. If the condition is false, either the statement following ELSE or OTHERWISE (statement-2) is executed, or, if the ELSE or OTHERWISE option is omitted, the next sentence is executed. If the condition is false and NEXT SENTENCE is written following ELSE, control passes explicitly to the next sentence.

When IF statements are not nested, statement-1 and statement-2 must represent imperative statements.

Nested IF Statements

The presence of one or more IF statements within the initial IF statement constitutes a "nested IF statement."

Statement-1 and statement-2 in IF statements may consist of one or more imperative statements and/or a conditional statement. If a conditional statement appears as statement-1 or as part of statement-1, it is said to be nested. Nesting statements is much like specifying subordinate arithmetic expressions enclosed in parentheses and combined in larger arithmetic expressions.

IF statements contained within IF statements must be considered as paired IF and ELSE combinations, proceeding from left to right. Thus, any ELSE encountered must be considered to apply to the immediately preceding IF that has not already been paired with an ELSE.

In the conditional statement in Figure 11, C stands for condition; S stands for any number of imperative statements; and the pairing of IF and ELSE is shown by the lines connecting them.

Chart 1 is a flowchart indicating the logical flow of the conditional statement in Figure 9.

IF1 C1 S1 IF2 C2 IF3 C3 S2 ELSE S3 ELSE S4 IF4 C4 IF5 C5 S5 ELSE S6

al – Statement-1 for IF1 (If C1 is false, the next sentence is executed, since there is no ELSE for it.)
b1 – Statement-1 for IF2
b2 – Statement-2 for IF2
c1 – Statement-1 for IF3
c2 – Statement-2 for IF3
d1 – Statement-1 for IF4 (If C4 is false, the next sentence is executed, since there is no ELSE for it.)
e1 – Statement-1 for IF5
e2 – Statement-2 for IF5

Figure 11.   Conditional Statements with Nested IF Statements

IF Statement

Chart 1. Logical Flow of Conditional Statement with Nested IF
Statements

```
        ****A1*********
        * EXECUTION OF *
        * IF STATEMENT *
        *    BEGINS    *
        ***************
               |
               v
            B1 *.*.
          *.      *.      FALSE
        *.    C1    .*---------------------------------------------------------------+
          *.      .*                                                                 |
            *.  .*                                                                   |
               * TRUE                                                                |
               |                                                                     |
               v                                                                     |
        ****C1*********                                                              |
        *             *                                                              |
        *     S1      *                                                              |
        *             *                                                              |
        ***************                                                              |
               |                                                                     |
               v                                                                     |
            D1 *.*.              ****D2*********          D3 *.*.                     |
          *.      *.    FALSE    *             *        *.      *.     FALSE          |
        *.    C2    .*--------->  *     S4      * ----->  *.    C4    .*------------->|
          *.      .*              *             *          *.      .*                 |
            *.  .*                ***************            *.  .*                   |
               * TRUE                                           * TRUE               |
               |                                                   |                 |
               v                                                   v                 |
            E1 *.*.              ****E2*********          E3 *.*.              ****E4*********
          *.      *.    FALSE    *             *        *.      *.     FALSE   *             *
        *.    C3    .*--------->  *     S3      * ----->  *.    C5    .*------->  *     S6      *
          *.      .*              *             *          *.      .*            *             *
            *.  .*                ***************            *.  .*              ***************
               * TRUE                  |                        * TRUE               |
               |                       |                        |                    |
               v                       |                        v                    |
        *****F1*********               |                  *****F3*********            |
        *             *               |                  *             *            |
        *     S2      *               |                  *     S5      *            |
        *             *               |                  *             *            |
        ***************               |                  ***************            |
               |                       |                        |                    |
               |                       v                        v                    v
               |<----------------------------------------------------------------<---+
               |
               v
        *****H1*********
        *             *
        * NEXT SENTENCE *
        *             *
        ***************
```
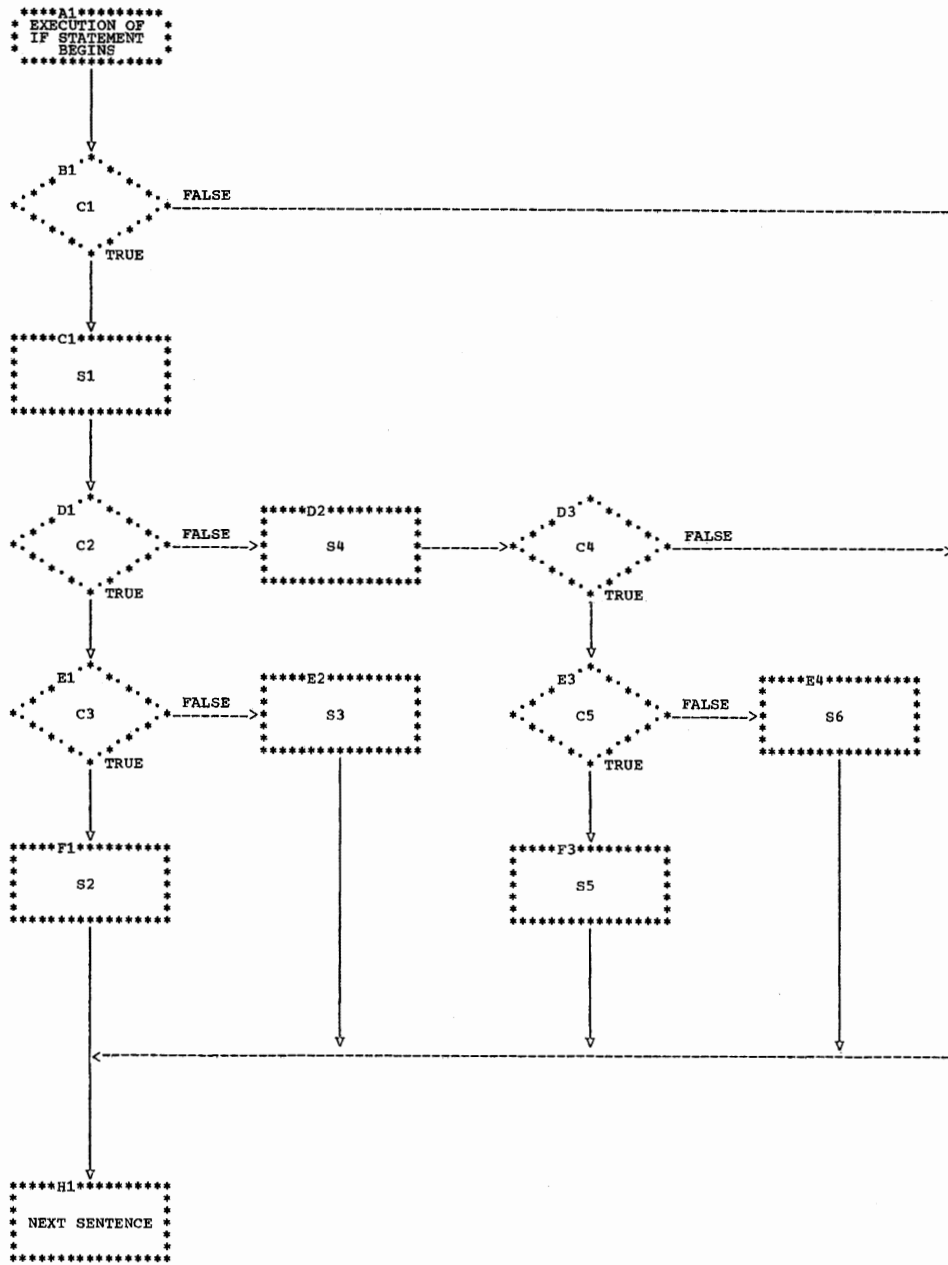
The Declaratives Section provides a method of including procedures that are invoked nonsynchronously: that is, they are executed not as part of the sequential coding written by the programmer, but rather when a condition occurs which cannot normally be tested by the programmer.

Although the system automatically handles checking and creation of standard labels and executes error recovery routines in the case of input/output errors, additional procedures may be specified by the COBOL programmer. The Report Writer feature also uses declarative procedures.

Since these procedures are executed only when labels of a file are to be processed, or at the time an error in reading or writing occurs or when a report group is to be produced, they cannot appear in the regular sequence of procedural statements. They must be written at the beginning of the Procedure Division in a subdivision called DECLARATIVES. A group of declarative procedures constitutes a declarative section. Related procedures are preceded by a USE sentence that specifies their function. A declarative section ends with the occurrence of another section-name with a USE sentence or with the words END DECLARATIVES.

The key words DECLARATIVES and END DECLARATIVES must each begin in Area A. No other text may appear on the same line.

```
+-----------------------------------------------------------------------+
|                           General Format                              |
+-----------------------------------------------------------------------+
|                                                                       |
|   PROCEDURE DIVISION.                                                  |
|                                                                       |
|   DECLARATIVES.                                                        |
|                                                                       |
|   {section-name SECTION.   USE sentence.                              |
|   {paragraph-name.   {sentence} ... } ... } ...                       |
|                                                                       |
|   END DECLARATIVES.                                                    |
|                                                                       |
+-----------------------------------------------------------------------+
```

The USE sentence identifies the type of declarative.

There are three formats of the USE sentence. Each is associated with one of the following types of procedures:

1. Input/output label handling

2. Input/output error-checking procedures

3. Report writing procedures

A USE sentence, when present, must immediately follow a section header in the Declarative portion of the Procedure Division and must be followed 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.

The USE sentence itself is never executed, rather it defines the conditions for the execution of the USE procedure.

LABEL PROCEDURE Declarative

Format 1 is used to provide user label-handling procedures.   There are two options of Format 1.

```
+----------------------------------------------------------------------+
|                              Format 1                                |
+----------------------------------------------------------------------+
|Option 1                                                              |
|                                                                      |
|        (BEFORE)                             [REEL]                    |
|   USE  {      }   STANDARD [BEGINNING]      |FILE|                    |
|        (AFTER )                             [UNIT]                    |
|                                                                      |
|                                    ({file-name} ...)                 |
|                                    ) OUTPUT        (                  |
|        LABEL PROCEDURE ON          ) INPUT        ( .                 |
|                                    ( I-O           )                 |
+----------------------------------------------------------------------+
|Option 2                                                              |
|                                                                      |
|        (BEFORE)                             [REEL]                    |
|   USE  {      }   STANDARD [ENDING]         |FILE|                    |
|        (AFTER )                             [UNIT]                    |
|                                                                      |
|                                    ({file-name} ...)                 |
|                                    ) OUTPUT        (                  |
|        LABEL PROCEDURE ON          ) INPUT        ( .                 |
|                                    ( I-O           )                 |
+----------------------------------------------------------------------+
```
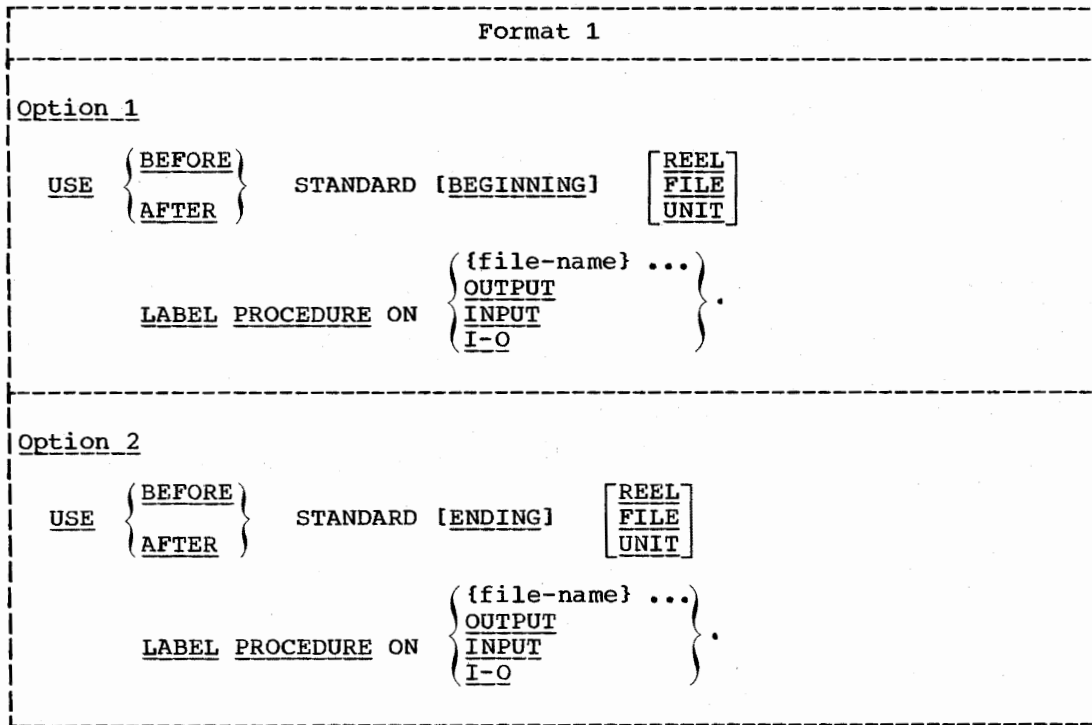
When BEFORE is specified, it indicates that nonstandard labels are to be processed.   Nonstandard labels may be specified only for tape files.

When AFTER is specified, it indicates that user labels follow standard file labels, and are to be processed.

Note: ASCII considerations for user label-handling procedures are given in Appendix E.

The labels must be listed as data-names in the LABEL RECORDS clause in the File Description entry for the file, and must be described as level-01 data items subordinate to the file entry.

If neither BEGINNING nor ENDING is specified, the designated procedures are executed for both beginning and ending labels.

If UNIT, REEL, or FILE are not included, the designated procedures are executed both for REEL or UNIT, whichever is appropriate, and for FILE labels.   The REEL option is not applicable to mass storage files. The UNIT option is not applicable to files in the random access mode since only FILE labels are processed in this mode.

The same file-name may appear in different specific arrangements of Format 1.   However, appearance of a file-name in a USE statement must not cause the simultaneous request for execution of more than one USE declarative.

If the _file-name_ option is used, the File Description entry for file-name must not specify a LABEL RECORDS ARE OMITTED clause.

The user label procedures are executed as follows when the OUTPUT, INPUT, or I-O options are specified:

- When OUTPUT is specified, only for files opened as output.

- When INPUT is specified, only for files opened as input.

- When I-O is specified, only for files opened as I-O.

The _file-name_ must not represent a sort-file.

If the INPUT, OUTPUT, or I-O option is specified, and an input, output, or input-output file, respectively, is described with a LABEL RECORDS ARE OMITTED clause, the USE procedures do not apply.

The standard system procedures are performed:

1. Before or after the user's beginning or ending input label check procedure is executed.

2. Before the user's beginning or ending output label is created.

3. After the user's beginning or ending output label is created, but before it is written on tape.

4. Before or after the user's beginning or ending input-output label check procedure is executed.

Within the procedures of a USE declarative in which the USE sentence specifies an option other than _file-name_, references to common label items need not be qualified by a file-name. A common label item is an elementary data item that appears in every label record of the program, but does not appear in any data record of this program. Such items must have identical descriptions and positions within each label record.

Within a Format 1 declarative section there must be no reference to any nondeclarative procedures. Conversely, in the nondeclarative portion there must be no reference to procedure-names that appear in the declaratives section, except that PERFORM statements may refer to a USE procedure, or to procedures associated with it.

The exit from a Format 1 declarative section is inserted by the compiler following the last statement in the section. All logical program paths within the section must lead to the exit point.

There is one exception: a special exit may be specified by the statement GO TO MORE-LABELS. When an exit is made from a Format 1 declarative section by means of this statement, the system will do one of the following:

1. Write the current beginning or ending label and then re-enter the USE section at its beginning for further creation of labels. After creating the last label, the user must exit by executing the last statement of the section.

2. Read an additional beginning or ending label, and then re-enter the USE section at its beginning for further checking of labels. When processing user labels, the section will be re-entered only if there is another user label to check. Hence, there need not be a program path that flows through the last statement in the section. For nonstandard labels, the compiler does not know how many labels exist. Therefore, the last statement in the section must be executed to terminate nonstandard label processing.

If a GO TO MORE-LABELS statement is not executed for a user label, the declarative section is not re-entered to check or create any immediately succeeding user labels.

When reading nonstandard header labels, it is the user's responsibility to read any tape marks that are used to terminate labels. The GO TO MORE-LABELS exit must be used and the declarative must recognize that a tape mark rather than data is being read. The final exit from the declarative must not be taken until the file is positioned just before the first data record.

No tape marks are written following nonstandard header labels. A tape mark is written following the last nonstandard trailer label on each reel.

SAMPLE LABEL DECLARATIVE PROGRAM

The following program creates two files, one with user labels, the other with nonstandard labels. To create the labels, the program contains a DECLARATIVES section, with USE procedures for creating both header and trailer labels.

The program illustrates the following items:

(1) For the two files requiring label creation, the LABEL RECORDS clause specifies the data-name option.

(2) The user labels are created by a USE AFTER BEGINNING/ENDING LABEL procedure.

(3) Two user header labels are to be created. Therefore the program utilizes the GO TO MORE-LABELS exit to create the second header label.

(4) The nonstandard labels are created by a USE BEFORE BEGINNING/ENDING LABEL procedure.

(5) Label information for the program is taken in part from the input file; therefore input records containing the information must be read and stored before the output files are opened, and the header label procedures invoked.

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. LABELPGM.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-360-F50.
       OBJECT-COMPUTER. IBM-360-F50.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT NO-LBL ASSIGN TO SYS010-UT-2400-S.
           SELECT USER ASSIGN TO SYS011-UT-2400-S.
           SELECT NON-STD ASSIGN TO SYS012-UT-2400-S.
       DATA DIVISION.
       FILE SECTION.
       FD   NO-LBL
            RECORD CONTAINS 80 CHARACTERS
            DATA RECORDS ARE IN-REC IN-LBL-HIST
            LABEL RECORD IS OMITTED.
       01   IN-REC.
            05   TYPEN PIC X(4).
                 88   NSTD VALUE 'NSTD'.
            05   DEPT-ID PIC X(11).
            05   BIL-PERIOD PIC X(5).
            05   NAME PIC X(20).
            05   AMOUNT PIC 9(6).
            05   FIL-NAM PIC X(15).
            05   SECUR-CODE PIC XX.
            05   AREAN PIC 9.
                 88   HDR-REC VALUE 9.
            05   ACCT-NUM PIC 9(10).
            05   SER-NUM PIC 9(6).
       01   IN-LBL-HIST.
            05   FILLER PIC X(4).
            05   FILE-HISTORY PIC X(76).
       FD   USER
            RECORD CONTAINS 80 CHARACTERS
            BLOCK CONTAINS 5 RECORDS
            DATA RECORD IS USR-REC
            LABEL RECORDS ARE USR-LBL USR-LBL-HIST.
(1)    01   USR-LBL.
            05   USR-HDR PIC X(4).
            05   DEPT-ID PIC X(11).
            05   USR-REC-CNT PIC 9(8) COMP-3.
            05   BIL-PERIOD PIC X(5).
            05   FILLER PIC X(53).
            05   SECUR-CODE PIC XX.
       01   USR-LBL-HIST.
            05   FILLER PIC X(4).
            05   LBL-HISTORY PIC X(76).
       01   USR-REC.
            05   TYPEN PIC X(4).
            05   FILLER PIC X(5).
            05   NAME PIC X(20).
            05   FILLER PIC X(4).
            05   ACCT-NUM PIC 9(10).
            05   AMOUNT PIC 9(6) COMP-3.
            05   FILLER PIC X(25).
            05   U-SFQ-NUMB PIC 9(8).
       FD   NON-STD
            RECORDING MODE IS U
            DATA RECORDS ARE NSTD-REC1 NSTD-REC2
            LABEL RECORD IS NSTD-LBL.
(1)    01   NSTD-LBL.
            05   NSTD-HDR PIC X(7).
            05   NSTD-REC-CNT PIC 9(8) COMP-3.
            05   FILLER PIC X(3).
            05   FIL-NAM PIC X(15).
            05   DEPT-ID PIC X(11).
            05   SER-NUM PIC 9(6).
```

```
        05   CREAT-DATE PIC X(8).
    01   NSTD-REC1.
        05   ACCT-NUM PIC 9(10).
        05   BIL-PERIOD PIC X(5).
        05   NAME PIC X(20).
        05   FILLER PIC X(40).
        05   AREAN PIC 9.
        05   FILLER PIC X(20).
        05   AMOUNT PIC 9(6) COMP-3.
    01   NSTD-REC2.
        05   ACCT-NUM PIC 9(10).
        05   BIL-PERIOD PIC X(5).
        05   NAME PIC X(20).
        05   FILLER PIC X(15).
        05   DEPT-ID PIC X(11).
        05   AMOUNT PIC 9(6).
        05   N-SEQ-NUMB PIC 9(8).
        05   AREAN PIC 9.
        05   FILLER PIC X(4).
    WORKING-STORAGE SECTION.
    77   N-REC-NUMB PIC 9(8) VALUE ZERO.
    77   U-REC-NUMB PIC 9(8) VALUE ZERO.
    77   SAV-FIL-NAM PIC X(15).
    77   SAV-DEPT-ID PIC X(11).
    77   LBL-SWITCH PIC 9 VALUE ZERO.
    77   USER-SWITCH PIC 9 VALUE ZERO.
    77   NSTD-SWITCH PIC 9 VALUE ZERO.
    77   NSTD-REC2-CNT PIC 9(8) VALUE ZERO.
    01   STOR-REC.
        05   DEPT-ID PIC X(11).
        05   BIL-PERIOD PIC X(5).
        05   SECUR-CODE PIC XX.
    PROCEDURE DIVISION.

    DECLARATIVES.
(2) USR-HDR-LBL SECTION. USE AFTER BEGINNING FILE
        LABEL PROCEDURE ON USER.
    A.   IF LBL-SWITCH = 0
            MOVE SPACES TO USR-LBL
            MOVE ZEROES TO USR-REC-CNT
            MOVE 'UHL1' TO USR-HDR
            MOVE CORRESPONDING STOR-REC TO USR-LBL
(3)         ADD 1 TO LBL-SWITCH   GO TO MORE-LABELS
               ELSE MOVE 'UHL2' TO USR-HDR
                    MOVE FILE-HISTORY TO LBL-HISTORY.
(2) USR-TRLR-LBL SECTION. USE AFTER ENDING FILE
        LABEL PROCEDURE ON USER.
    B.   MOVE SPACES TO USR-LBL.
        MOVE 'UTL1' TO USR-HDR.
        MOVE SAV-DEPT-ID TO DEPT-ID IN USR-LBL.
        MOVE U-REC-NUMB TO USR-REC-CNT,
(4) NSTD-HDR-LBL SECTION. USE BEFORE BEGINNING FILE
        LABEL PROCEDURE ON NON-STD.
    C.   MOVE 'NSTHDR1' TO NSTD-HDR.
        MOVE ZEROES TO NSTD-REC-CNT
        MOVE CORRESPONDING IN-REC TO NSTD-LBL.
        MOVE CURRENT-DATE TO CREAT-DATE.
        MOVE FIL-NAM OF IN-RFC TO SAV-FIL-NAM.
(4) NSTD-TRLR-LBL SECTION. USE BEFORE ENDING FILE
        LABEL PROCEDURE ON NON-STD.
    D.   MOVE SPACES TO NSTD-LBL.
        MOVE 'NSTEOF ' TO NSTD-HDR.
        MOVE N-REC-NUMB TO NSTD-REC-CNT.
        MOVE SAV-FIL-NAM TO FIL-NAM IN NSTD-LBL.
    END DECLARATIVES.

    NON-DECLARATIVE SECTION.
        OPEN INPUT NO-LBL.
```

```
      READ-IN.
            READ NO-LBL AT END GO TO END-JOB.
      E.    IF NSTD NEXT SENTENCE
                  ELSE GO TO PROCESS-USER.
            IF NSTD-SWITCH = 1 NEXT SENTENCE
                  ELSE ADD 1 TO NSTD-SWITCH
                        OPEN OUTPUT NON-STD
                        GO TO READ-IN.
            ADD 1 TO N-REC-NUMB.
            IF HDR-REC MOVE SPACES TO NSTD-REC1
                        MOVE CORRESPONDING IN-REC TO NSTD-REC1
                        WRITE NSTD-REC1
                  ELSE ADD 1 TO NSTD-REC2-CNT
                        MOVE SPACES TO NSTD-REC2
                        MOVE CORRESPONDING IN-REC TO NSTD-REC2
                        MOVE NSTD-REC2-CNT TO N-SEQ-NUMB
                        WRITE NSTD-REC2.
            GO TO READ-IN.
      PROCESS-USER.
            IF USER-SWITCH = 1 NEXT SENTENCE
                  ELSE ADD 1 TO USER-SWITCH
                        MOVE CORRESPONDING IN-REC TO STOR-REC
                        MOVE DEPT-ID OF IN-REC TO SAV-DEPT-ID
            PERFORM READ-IN
                        OPEN OUTPUT USER
                        GO TO READ-IN.
            ADD 1 TO U-REC-NUM3.
            MOVE CORRESPONDING IN-REC TO USR-REC.
            MOVE U-REC-NUMB TO U-SEQ-NUMB
            WRITE USR-REC
            GO TO READ-IN.
      END-JOB.
            CLOSE NO-LBL.
            IF USER-SWITCH = 1 CLOSE USER.
            IF NSTD-SWITCH = 1 CLOSE NON-STD.
            STOP RUN.
```

A Format 2 USE sentence specifies procedures to be followed if an input/output error occurs during file processing.

```
r---------------------------------------------------------------------------
|                                 Format 2                                  |
|--------------------------------------------------------------------------|
|                                                                          |
|   USE AFTER STANDARD ERROR PROCEDURE                                      |
|                                                                          |
|           / {file-name-1} ...                                          \  |
|          /  file-name-2 GIVING data-name-1 [data-name-2]  \              |
|      ON <   INPUT                                           >    .        |
|          \  OUTPUT                                         /              |
|           \ I-O                                          /                |
|                                                                          |
L--------------------------------------------------------------------------J
```

When Format 2 is used, automatic system error routines are executed before user-specified procedures.

USE declaratives which specify error handling procedures are activated when an input/output error occurs during execution of a READ, WRITE, REWRITE, or START statement.

Within the section, the file associated with the USE sentence may not be referred to by an OPEN, START, SEEK, READ, WRITE, or REWRITE statement.

Within a USE procedure there must be no reference to nondeclarative procedures except when an exit is taken with a GO TO statement. Conversely, in the nondeclarative portion, there must be no reference to procedure-names that appear in the declaratives portion, except that PERFORM statements may refer to a USE declarative, or to procedures associated with such a declarative.

When either the file-name-1 or file-name-2 option is used, user error handling procedures are executed for input/output errors occurring during execution of a READ, WRITE, REWRITE, or START statement for that file only.

A file-name must not be referred to, implicitly or explicitly, in more than one Format 2 USE sentence.

The user error procedures are executed when the INPUT, OUTPUT, or I-O options are specified and an input/output error occurs, as follows:

- When INPUT is specified, only for files opened as INPUT.

- When OUTPUT is specified, only for files opened as OUTPUT.

- When I-O is specified, only for files opened as I-O (input-output).

When the GIVING option is used, data-name-1 will be set to reflect the error condition that caused the error declarative to be entered. A value of 1 in an error byte indicates that its corresponding error condition has occurred. Individual bytes will correspond to specific error conditions depending on the type of file being processed as shown in Figure 12.

Error Byte	INDEXED SEQUENTIAL	DIRECT	SEQUENTIAL
1	DASD error	Data check in count	Parity error
2	Wrong length record	Wrong length record	Wrong length record
3	Prime data full	No room found	-
4	Cylinder index full	Data check in key or data	-
5	Master index full	-	-
6	Overflow area full	-	-
7	-	-	-
8	-	-	-

Figure 12. Error Byte Meaning for the GIVING Option of an Error Declarative

Data-name-1 must be an 8-byte external-decimal item. It must be defined in the Working-Storage Section.

Data-name-2, if specified, will contain the block in error (or blank, if no data was transmitted). Data-name-2 will contain the block in error only when the error occurs during a READ operation.

Data-name-2 must be large enough to hold the largest physical block that exists or will be written on file-name-2. It must be defined in the Working-Storage or Linkage Section. If data-name-2 is defined in the Linkage Section, the block in error is referenced in the buffer area; no storage need be defined for the error block.

An exit from this type of declarative section can be effected by executing the last statement in the section (normal return), or by means of a GO TO statement. Table 14 summarizes the facilities and limitations associated with each file-processing technique when an error occurs.

The normal return from an error declarative is to the statement following the input/output statement that caused the error.

Table 14.   File Processing Techniques and Associated Error Declaratives Capabilities

Access	Organi-zation	Type of I/O Statement	Error Bytes	Error Declarative Written		No Error Declarative Written
				Normal Return	GO TO Exit	
SEQUEN-TIAL (or not speci-fied)	Sequen-tial	READ	1 or 2	Continued process-ing of file per-mitted; bad block is bypassed	User limited to CLOSE for file	Diagnostic error message is printed; job is terminated
		WRITE	1 or 2	Continued process-ing of file per-mitted; bad block has been written	Continued process-ing of file per-mitted; bad block has been written	
	Direct	READ	1, 2, or 4	User limited to CLOSE for file	User limited to CLOSE for file	
RANDOM	Direct	READ	1, 2, or 4	Continued process-ing of file per-mitted; bad block has not been by-passed	Continued process-ing of file per-mitted; bad block has not been by-passed	
		WRITE	1, 2, 3, or 4	Continued process-ing of file per-mitted; bad block has been written	Continued process-ing of file per-mitted; bad block has been written	
		REWRITE	1, 2, or 4			
SEQUEN-TIAL (or not speci-fied)	Indexed	READ	1 or 2	Continued process-ing of file per-mitted; bad block has not been by-passed	Continued process-ing of file per-mitted; bad block has not been by-passed	
		START	1	Continued process-ing of file per-mitted	Continued process-ing of file per-mitted	
		REWRITE	1 or 2	Continued process-ing of file per-mitted; bad block has been written	Continued process-ing of file per-mitted; bad block has been written	
		WRITE	1, 2, 3, 4, or 5	User limited to CLOSE for file	User limited to CLOSE for file	
RANDOM	Indexed	READ	1 or 2	Continued process-ing of file per-mitted; bad block has not been by-passed	Continued process-ing of file per-mitted; bad block has not been by-passed	
		REWRITE	1 or 2	Continued process-ing of file per-mitted; bad block has been written	Continued process-ing of file per-mitted; bad block has been written	
		WRITE	1, 2, or 6	User limited to CLOSE for file	User limited to CLOSE for file	

A Format 3 USE sentence specifies Procedure Division statements that are executed just before a report group named in the Report Section of the Data Division is produced (see "Report Writer").

```
|--------------------------------------------------------------------|
|                            Format 3                                |
|--------------------------------------------------------------------|
|                                                                    |
|   USE BEFORE REPORTING data-name.                                  |
|                                                                    |
|--------------------------------------------------------------------|
```

ARITHMETIC STATEMENTS

The arithmetic statements are used for computations. Individual
operations are specified by the ADD, SUBTRACT, MULTIPLY, and DIVIDE
statements. These operations can be combined symbolically in a formula,
using the COMPUTE statement.

Because there are several options common to the arithmetic state-
ments, their discussion precedes individual statement descriptions.

CORRESPONDING Option

The CORRESPONDING option enables computations to be performed on
elementary items of the same name simply by specifying the group item to
which they belong. The word CORRESPONDING may be abbreviated as CORR.

Both identifiers following CORRESPONDING must refer to group items.
For the purposes of this discussion, these identifiers will be called d_1
and d_2.

Elementary data items from each group are considered CORRESPONDING
when both data items have the same name and qualification, up to but not
including d_1 and d_2.

Neither d_1 nor d_2 may be a data item with level number 66, 77, or 88,
nor may either be described with the USAGE IS INDEX clause. Neither d_1
nor d_2 may be a FILLER item.

Each data item subordinate to d_1 or d_2 that is described with a
REDEFINES, OCCURS, or USAGE IS INDEX clause is ignored; any items
subordinate to such data items are also ignored. However, d_1 or d_2 may
themselves be described with REDEFINES or OCCURS clauses, or be
subordinate to items described with REDEFINES or OCCURS clauses.

Each FILLER item subordinate to d_1 or d_2 is ignored. However, d_1 or
d_2 may be subordinate to a FILLER item.

GIVING Option

If the GIVING option is specified, the value of the identifier that
follows the word GIVING is set equal to the calculated result of the
arithmetic operation. This identifier, since not itself involved in the
computation, may be a numeric edited item.

ROUNDED Option

After decimal point alignment, the number of places in the fraction
of the result of an arithmetic operation is compared with the number of
places provided for the fraction of the resultant identifier.

When the size of the fractional result exceeds the number of places provided for its storage, truncation occurs unless ROUNDED is specified. When ROUNDED is specified, the least significant digit of the resultant identifier has its value increased by 1 whenever the most significant digit of the excess is greater than or equal to 5.

When the resultant identifier is described by a PICTURE clause containing P's and when the number of places in the calculated result exceeds this size, rounding or truncation occurs relative to the rightmost integer position for which storage is allocated.

SIZE ERROR Option

If, after decimal point alignment, the value of a result exceeds the largest value that can be contained in the associated resultant identifier, a size error condition exists. Division by zero always causes a size error condition. The size error condition applies only to the final results of an arithmetic operation and does not apply to intermediate results. If the ROUNDED option is specified, rounding takes place before checking for size error. When such a size error condition occurs, the subsequent action depends on whether or not the SIZE ERROR option is specified.

If the SIZE ERROR option is not specified and a size error condition occurs, the value of the resultant identifier affected may be unpredictable.

If the SIZE ERROR option is specified and a size error condition occurs, the value of the resultant identifier affected by the size error is not altered. After completion of the execution of the arithmetic operation, the imperative statement in the SIZE ERROR option is executed.

For COMPUTATIONAL-1 and COMPUTATIONAL-2 items, only division by zero causes the imperative-statement in the SIZE ERROR option to be executed.

Overlapping Operands

When the sending and receiving operands of an arithmetic statement or a MOVE statement share a part of their storage (that is, when the operands overlap), the result of the execution of such a statement is unpredictable.

ADD Statement

The ADD statement causes two or more numeric operands to be summed and the result to be stored.

```
+-----------------------------------------------------------------------+
|                             Format 1                                  |
+-----------------------------------------------------------------------+
|                                                                       |
|        ( identifier-1 )   [ identifier-2 ]                            |
|  ADD  {              }   [             ]  ... TO identifier-m [ROUNDED]|
|        ( literal-1    )   [ literal-2    ]                            |
|                                                                       |
|     [identifier-n [ROUNDED]] ... [ON SIZE ERROR imperative-statement] |
|                                                                       |
+-----------------------------------------------------------------------+
```

```
+-----------------------------------------------------------------------+
|                             Format 2                                  |
+-----------------------------------------------------------------------+
|                                                                       |
|        ( identifier-1)   ( identifier-2)   [ identifier-3]            |
|  ADD  {             }   {             }   [            ]   ...        |
|        ( literal-1   )   ( literal-2   )   [ literal-3    ]            |
|                                                                       |
|     GIVING identifier-m [ROUNDED] [ON SIZE ERROR imperative-statement]|
|                                                                       |
+-----------------------------------------------------------------------+
```

```
+-----------------------------------------------------------------------+
|                             Format 3                                  |
+-----------------------------------------------------------------------+
|                                                                       |
|        ( CORR          )                                             |
|  ADD  {              }  identifier-1 TO identifier-2                  |
|        ( CORRESPONDING )                                             |
|                                                                       |
|     [ROUNDED] [ON SIZE ERROR imperative-statement]                   |
|                                                                       |
+-----------------------------------------------------------------------+
```

Format 1 -- the values of the operands preceding the word TO are added together, and the sum is added to the current value of identifier-m (identifier-n), etc. The result is stored in identifier-m (identifier-n), etc.

Format 2 -- when the GIVING option is used, there must be at least two operands preceding the word GIVING. The values of these operands are added together, and the sum is stored as the new value of identifier-m.

In Formats 1 and 2 each identifier must refer to an elementary numeric item, with the exception of identifiers appearing to the right of the word GIVING. These may refer to numeric edited data items.

Each literal must be a numeric literal.

The maximum size of each operand is 18 decimal digits. The maximum size of the resulting sum, after decimal point alignment, is 18 decimal digits.

Format 3 -- when the CORRESPONDING option is used, elementary data items within identifier-1 are added to and stored in corresponding elementary data items within identifier-2. Identifier-1 and identifier-2 must be group items.

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

COMPUTE Statement

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

```
┌─────────────────────────────────────────────────────────────────┐
│                              Format                              │
├─────────────────────────────────────────────────────────────────┤
│                                                                  │
│                                   ⎧ identifier-2          ⎫      │
│   COMPUTE identifier-1 [ROUNDED] = ⎨ literal-1             ⎬      │
│                                   ⎩ arithmetic-expression ⎭      │
│                                                                  │
│       [ON SIZE ERROR imperative-statement]                       │
│                                                                  │
└─────────────────────────────────────────────────────────────────┘
```

Literal-1 must be a numeric literal.

Identifier-2 must refer to an elementary numeric item. Identifier-1 may describe a numeric edited data item.

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

The arithmetic-expression option permits the use of a meaningful combination of identifiers, numeric literals, and arithmetic operators. Hence, the user can combine arithmetic operations without the restrictions imposed by the arithmetic statements ADD, SUBTRACT, MULTIPLY, and DIVIDE.

As in all arithmetic statements, the maximum size of each operand is 18 decimal digits.

DIVIDE Statement

The DIVIDE statement is used to find the quotient resulting from the division of one data item into another data item.

```
+----------------------------------------------------------------------+
|                              Format 1                                |
+----------------------------------------------------------------------+
|                                                                      |
|              ( identifier-1 )                                        |
|   DIVIDE     {             }  INTO identifier-2   [ROUNDED]          |
|              ( literal-1    )                                        |
|                                                                      |
|         [ON SIZE ERROR imperative-statement]                        |
|                                                                      |
+----------------------------------------------------------------------+
```

```
+----------------------------------------------------------------------+
|                              Format 2                                |
+----------------------------------------------------------------------+
|                                                                      |
|              ( identifier-1 ) ( INTO ) ( identifier-2 )             |
|   DIVIDE     {             } {      } {             }  GIVING identifier-3 |
|              ( literal-1    ) ( BY   ) ( literal-2    )             |
|                                                                      |
|         [ROUNDED]  [REMAINDER identifier-4]                         |
|                                                                      |
|         [ON SIZE ERROR imperative-statement]                       |
|                                                                      |
+----------------------------------------------------------------------+
```

When Format 1 is used, the value of identifier-1 (or literal-1) is divided into the value of identifier-2. The value of the dividend (identifier-2) is replaced by the value of the quotient.

When Format 2 is used, the value of identifier-1 (or literal-1) is divided into or by identifier-2 (or literal-2), the quotient is stored in identifier-3, and the remainder optionally is stored in identifier-4.

A remainder is defined as the result of subtracting the product of the quotient and the divisor from the dividend. When the REMAINDER option is specified, none of the identifiers may refer to floating-point items. If the ROUNDED option is also specified, the quotient is rounded after the remainder is determined.

Each identifier must refer to an elementary numeric item except the identifier following the word GIVING, which may be a numeric edited item.

Each literal must be a numeric literal.

The maximum size of the resulting quotient, after decimal point alignment, is 18 decimal digits. The maximum size of the resulting remainder (if specified), after decimal point alignment, is 18 decimal digits.

Division by zero always results in a size error condition.

MULTIPLY Statement

The MULTIPLY statement is used to multiply one data item by another data item.

```
-------------------------------------------------------------------------
|                                Format 1                                |
|-----------------------------------------------------------------------|
|                                                                        |
|           ( identifier-1 )                                             |
|  MULTIPLY <              >  BY identifier-2  [ROUNDED]                  |
|           ( literal-1    )                                             |
|                                                                        |
|      [ON SIZE ERROR imperative-statement]                              |
|                                                                        |
-------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------
|                                Format 2                                |
|-----------------------------------------------------------------------|
|                                                                        |
|           ( identifier-1 )       ( identifier-2 )                      |
|  MULTIPLY <              >  BY   <              >  GIVING identifier-3   |
|           ( literal-1    )       ( literal-2    )                      |
|                                                                        |
|      [ROUNDED]    [ON SIZE ERROR imperative-statement]                 |
|                                                                        |
-------------------------------------------------------------------------
```

When Format 1 is used, the value of identifier-1 (or literal-1) is multiplied by the value of identifier-2. The value of the multiplier (identifier-2) is replaced by the product.

When Format 2 is used, the value of identifier-1 (or literal-1) is multiplied by identifier-2 (or literal-2), and the product is stored in identifier-3.

Each identifier must refer to an elementary numeric item except the identifier following the word GIVING, which may be a numeric edited item.

Each literal must be a numeric literal.

The maximum size of each opernad is 18 decimal digits. The maximum size of the resulting product, after decimal point alignment, is 18 decimal digits.

SUBTRACT Statement

The SUBTRACT statement is used to subtract one, or the sum of two or more, numeric data items from another data item(s).

```
┌─────────────────────────────────────────────────────────────────────┐
│                              Format 1                               │
├─────────────────────────────────────────────────────────────────────┤
│                                                                     │
│              ⎧ identifier-1 ⎫  ⎡ identifier-2 ⎤                      │
│   SUBTRACT   ⎨              ⎬  ⎢              ⎥   ...                │
│              ⎩ literal-1    ⎭  ⎣ literal-2    ⎦                      │
│                                                                     │
│      FROM identifier-m    [ROUNDED]                                 │
│                                                                     │
│      [identifier-n [ROUNDED]] ... [ON SIZE ERROR imperative-statement]│
│                                                                     │
└─────────────────────────────────────────────────────────────────────┘
```

```
┌─────────────────────────────────────────────────────────────────────┐
│                              Format 2                               │
├─────────────────────────────────────────────────────────────────────┤
│                                                                     │
│              ⎧ identifier-1 ⎫  ⎡ identifier-2 ⎤                      │
│   SUBTRACT   ⎨              ⎬  ⎢              ⎥   ...                │
│              ⎩ literal-1    ⎭  ⎣ literal-2    ⎦                      │
│                                                                     │
│              ⎧ identifier-m ⎫                                       │
│   FROM       ⎨              ⎬  GIVING identifier-n                   │
│              ⎩ literal-m    ⎭                                       │
│                                                                     │
│   [ROUNDED]    [ON SIZE ERROR imperative-statement]                 │
│                                                                     │
└─────────────────────────────────────────────────────────────────────┘
```

```
┌─────────────────────────────────────────────────────────────────────┐
│                              Format 3                               │
├─────────────────────────────────────────────────────────────────────┤
│                                                                     │
│              ⎧ CORR          ⎫                                      │
│   SUBTRACT   ⎨               ⎬  identifier-1 FROM identifier-2       │
│              ⎩ CORRESPONDING ⎭                                      │
│                                                                     │
│   [ROUNDED] [ON SIZE ERROR imperative-statement]                    │
│                                                                     │
└─────────────────────────────────────────────────────────────────────┘
```

Format 1 -- all literals or identifiers preceding the word FROM are added together, and this total is subtracted from identifier-m and identifier-n (if stated), etc. The result of the subtraction is stored as the new value of identifier-m, identifier-n, etc.

Format-2 -- all literals or identifiers preceding the word FROM are added together, and this total is subtracted from literal-m or identifier-m. The result of the subtraction is stored as the value of identifier-n.

Format 3 -- data items in identifier-1 are subtracted from, and the difference stored into corresponding data items in identifier-2. When the CORRESPONDING option is used in conjunction with ON SIZE ERROR and an ON SIZE ERROR condition arises, the result for SUBTRACT is analogous to that for ADD.

Each <u>identifier</u> must refer to an elementary numeric item except the identifier following the word GIVING, which may be a numeric edited item.

Each <u>literal</u> must be a numeric literal.

The maximum size of each operand is 18 decimal digits. The maximum size of the resulting difference, after decimal point alignment, is 18 decimal digits.

PROCEDURE BRANCHING STATEMENTS

Statements, sentences, and paragraphs in the Procedure Division are ordinarily executed sequentially. The procedure branching statements (GO TO, ALTER, PERFORM, STOP, and EXIT) allow alterations in the sequence.

GO TO Statement

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

```
r------------------------------------------------------------------¬
|                            Format 1                              |
|------------------------------------------------------------------|
|                                                                  |
|     GO TO procedure-name-1                                        |
|                                                                  |
L------------------------------------------------------------------+
```

```
r------------------------------------------------------------------¬
|                            Format 2                              |
|------------------------------------------------------------------|
|                                                                  |
|     GO TO procedure-name-1 [procedure-name-2] ...                 |
|                                                                  |
|         DEPENDING ON identifier                                   |
|                                                                  |
L------------------------------------------------------------------+
```

```
r------------------------------------------------------------------¬
|                            Format 3                              |
|------------------------------------------------------------------|
|                                                                  |
|     GO TO.                                                        |
|                                                                  |
L------------------------------------------------------------------+
```

When Format 1 is specified, control is passed to procedure-name-1 or to another procedure name if the GO TO statement has been changed by an ALTER statement. (If the latter is the case, the GO TO statement must have a paragraph name, and the GO TO statement must be the only statement in the paragraph.)

If a GO TO statement represented by Format 1 appears in an imperative sentence, it must appear as the only or last statement in a sequence of imperative statements.

When Format 2 is used, control is transferred to one of a series of procedures, depending on the value of the identifier. When identifier has a value of 1, control is passed to procedure-name-1; a value of 2 causes control to be passed to procedure-name-2, ...; a value of n causes control to be passed to procedure-name-n. For the GO TO statement to have effect, identifier must represent a positive or unsigned integer, i.e., 1, 2, ..., n. If the value of the identifier is anything other than a value within the range 1 through n, the GO TO statement is ignored. The number of procedure-names must not exceed 2031.

Identifier is the name of a numeric elementary item described as an integer. Its PICTURE must be of four digits or less. Its USAGE must be DISPLAY, COMPUTATIONAL, or COMPUTATIONAL-3.

When Format 3 is used, an ALTER statement, referring to the GO TO statement, must have been executed prior to the execution of the GO TO statement. The GO TO statement must immediately follow a paragraph name and must be the only statement in the paragraph.

ALTER Statement

The ALTER statement is used to change the transfer point specified in a GO TO statement.

```
-------------------------------------------------------------------------
|                                 Format                                 |
|-----------------------------------------------------------------------|
|                                                                       |
|   ALTER procedure-name-1 TO [PROCEED TO] procedure-name-2             |
|                                                                       |
|                                                                       |
|        [procedure-name-3 TO [PROCEED TO] procedure-name-4]...         |
|                                                                       |
-------------------------------------------------------------------------
```

Procedure-name-1, procedure-name-3, etc., must be the names of paragraphs that contain only one sentence consisting of a GO TO statement without the DEPENDING option.

Procedure-name-2, procedure-name-4, etc., must be the names of paragraphs or sections in the Procedure Division.

The effect of the ALTER statement is to replace the procedure-name operands of the GO TO statements with procedure-name-2, procedure-name-4, etc., of the ALTER statement, where the paragraph-name containing the GO TO statement is procedure-name-1, procedure-name-3, etc. For example:

```
    PARAGRAPH-1.
        GO TO BYPASS-PARAGRAPH.
    PARAGRAPH-1A.
            .
            .
            .
    BYPASS-PARAGRAPH.
            .
            .
            .
        ALTER PARAGRAPH-1 TO PROCEED TO PARAGRAPH-2.
            .
            .
            .
    PARAGRAPH-2.
            .
            .
            .
```

Before the ALTER statement is executed, when control reaches PARAGRAPH-1, the GO TO statement transfers control to BYPASS-PARAGRAPH. After execution of the ALTER statement, however, when control reaches PARAGRAPH-1, the GO TO statement transfers control to PARAGRAPH-2.

Segmentation Information: 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. All other uses of the ALTER statement are valid and are performed even if the GO TO to which the ALTER refers is in an overlayable fixed segment (see "Segmentation").

PERFORM Statement

The PERFORM statement is used to depart from the normal sequence of procedures in order to execute a statement, or a series of statements, a specified number of times; or until a predetermined condition is satisfied. After the statements are executed, control is returned to the statement after the PERFORM statement.

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

```
-----------------------------------------------------------------------
|                             Format 2                                |
|---------------------------------------------------------------------|
|                                                                     |
|  PERFORM procedure-name-1 [THRU procedure-name-2]                   |
|                                                                     |
|        ( identifier-1 )                                             |
|        {             }  TIMES                                       |
|        ( integer-1    )                                             |
|                                                                     |
-----------------------------------------------------------------------
```

```
-----------------------------------------------------------------------
|                             Format 3                                |
|---------------------------------------------------------------------|
|                                                                     |
|  PERFORM procedure-name-1 [THRU procedure-name-2]                   |
|                                                                     |
|        UNTIL condition-1                                            |
|                                                                     |
-----------------------------------------------------------------------
```

```
-----------------------------------------------------------------------
|                             Format 4                                |
|---------------------------------------------------------------------|
|                                                                     |
|  PERFORM procedure-name-1 [THRU procedure-name-2]                   |
|                                                                     |
|              ( index-name-1 )        ( index-name-2 )              |
|      VARYING {             }  FROM  { literal-2     }              |
|              ( identifier-1 )        ( identifier-2 )              |
|                                                                     |
|           ( literal-3    )                                         |
|      BY   {             }  UNTIL  condition-1                      |
|           ( identifier-3 )                                         |
|                                                                     |
|              ( index-name-4 )        ( index-name-5 )              |
|      [AFTER {             }  FROM  { literal-5     }              |
|              ( identifier-4 )        ( identifier-5 )              |
|                                                                     |
|           ( literal-6    )                                         |
|      BY   {             }  UNTIL condition-2                       |
|           ( identifier-6 )                                         |
|                                                                     |
|              ( index-name-7 )        ( index-name-8 )              |
|      [AFTER {             }  FROM  { literal-8     }              |
|              ( identifier-7 )        ( identifier-8 )              |
|                                                                     |
|           ( literal-9    )                                         |
|      BY   {             }  UNTIL condition-3]]                     |
|           ( identifier-9 )                                         |
|                                                                     |
-----------------------------------------------------------------------
```

Each procedure-name must be the name of a section or paragraph in the Procedure Division.

Each identifier represents a numeric elementary item described in the Data Division. In Format 2, and Format 4 with the AFTER option, each identifier represents a numeric item described as an integer. However, when Format 4 with the AFTER option is used, this compiler allows each identifier to be described as a non-integral numeric item.

Each literal represents a numeric literal.

Whenever a PERFORM statement is executed, control is transferred to the first statement of the procedure named procedure-name-1. Control is always returned to the statement following the PERFORM statement. The point from which this control is passed is determined as follows:

1. If procedure-name-1 is a paragraph name and procedure-name-2 is not specified, the return is made after the execution of 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 made after the execution of the last sentence of the last paragraph in procedure-name-1.

3. If procedure-name-2 is specified and it is a paragraph name, the return is made after the execution of the last statement of that paragraph.

4. If procedure-name-2 is specified and it is a section name, the return is made after the excecution of the last sentence of the last paragraph in the section.

When both procedure-name-1 and procedure-name-2 are specified, GO TO and PERFORM statements may appear within the sequence of statements within these paragraphs or sections. When procedure-name-1 alone is specified, PERFORM statements may appear within the procedure. GO TO may also appear but may not refer to a procedure-name outside the range of procedure-name-1.

When a PERFORM statement includes within its range another PERFORM, this embedded PERFORM statement must have its range either totally included in or excluded from the range of the original PERFORM statement. That is, the exit point of the first PERFORM cannot be contained within the range of the second PERFORM, except as a common exit point. Embedded PERFORM or GO TO statements may have their exit point at the same point that the original PERFORM makes its exit. This common exit point must be the name of a paragraph consisting solely of an EXIT statement.

Control may be passed to a sequence of statements that lies between the entry and exit points of a PERFORM statement by means other than a PERFORM. In this case, control passes through the last statement of the procedure to the following statement as if no PERFORM statement referred to these procedures.

FORMAT 1: When Format 1 is used, the procedure(s) referred to are executed once, and control returns to the statement following the PERFORM statement.

FORMAT 2: When Format 2 is used, the procedure(s) are performed the number of times specified by identifier-1 or integer-1. Once the TIMES option is satisfied, control is transferred to the statement following the PERFORM statement.

The following rules apply to the use of a Format 2 PERFORM statement:

1. If integer-1 or identifier-1 is zero or a negative number at the time the PERFORM statement is initiated, control passes to the statement following the PERFORM statement.

2. Once the PERFORM statement has been initiated, any reference to identifier-1 has no effect in varying the number of times the procedures are initiated.

FORMAT 3: When Format 3 is used, the specified procedures are performed until the condition specified by the UNTIL option is true. At this time, control is transferred to the statement following the PERFORM statement. If the condition is true at the time that the PERFORM statement is encountered, the specified procedure(s) are not executed.

FORMAT 4: Format 4 is used to augment the value of one or more identifiers or index-names during the execution of a PERFORM statement.

When executing a Format 4 PERFORM statement, the initial values of identifier-2 (index-name-2) and identifier-5 (index-name-5) must be positive in order to conform with the standard. However, this compiler allows these initial values to be negative.

In the following discussion of Format 4, every reference to identifier-n also refers to index-name-n except when identifier-n is the object of the BY option. Also, when index-names are used, the FROM and BY clauses have the same effect as in a SET statement (see "Table Handling").

During execution of the PERFORM statement, reference to index-names or identifiers of the FROM option has no effect in altering the number of times the procedures are to be executed. Changing the value of index-names or identifiers of the VARYING option or identifiers of the BY option, however, will change the number of times the procedures are executed.

When one identifier is varied, the following is the sequence of events:

1. Identifier-1 is set equal to its starting value, identifier-2 or literal-2.

2. If condition-1 is false, the specified procedure(s) are executed once.

3. The value of identifier-1 is augmented by the specified increment or decrement, identifier-3 or literal-3, and condition-1 is evaluated again.

4. Steps 2 and 3 are repeated, if necessary, until the condition is true. When the condition is true, control passes directly to the statement following the PERFORM statement. If the condition is true for the starting value of identifier-1, the procedure(s) are not executed, and control passes directly to the statement following the PERFORM statement.

Chart 2 is a flowchart illustrating the logic of the PERFORM statement when one identifier is varied.

When two identifiers are varied, the following is the sequence of events:

1. Identifier-1 and identifier-4 are set to their initial values, identifier-2 (or literal-2) and identifier-5 (or literal-5), respectively.

2. Condition-1 is evaluated; if true, control is passed to the statement following the PERFORM statement; if false, condition-2 is evaluated.

3. If condition-2 is false, procedure-name-1 through procedure-name-2 (if specified) is executed once.

4. Identifier-4 is augmented by identifier-6 (or literal-6), and condition-2 is evaluated again.

5. If condition-2 is false, steps 3 and 4 are repeated.

6. If condition-2 is true, identifier-4 is set to its initial value, identifier-5.

7. Identifier-1 is augmented by identifier-3 (or literal-3).

8. Steps 2 through 7 are repeated until condition-1 is true.

At the termination of the PERFORM statement, if condition-1 was true when the PERFORM statement was encountered, identifier-1 and identifier-4 contain their initial values. Otherwise, identifier-1 has a value that differs from its last used setting by an increment or S1S1decrement, as the case may be.

Chart 3 is a flowchart illustrating the logic of the PERFORM statement when two identifiers are varied.

For three identifiers, the mechanism is the same as for two identifiers except that identifier-7 goes through the complete cycle each time that identifier-4 is augmented by identifier-6 or literal-6, which in turn goes through a complete cycle each time identifier-1 is varied.

Chart 4 is a flowchart illustrating the logic of the PERFORM statement when three identifiers are varied.

SEGMENTATION INFORMATION: A PERFORM statement appearing in a section whose priority is less than the segment limit can have within its range only one of the following:

1. Sections each of which has a priority number less than 50

2. Sections wholly contained in a single segment whose priority number is greater than 49

However, this compiler allows the PERFORM to have within its range sections with any priority numbers.

A PERFORM statement appearing in a section whose priority number is equal to or greater than the segment limit can have within its range only one of the following:

1. Sections each of which has the same priority number as that containing the PERFORM statement

2. Sections with a priority number less than the segment limit

However, this compiler allows the PERFORM to have within its range sections with any priority numbers.

When a procedure-name is 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 for each execution of the PERFORM statement (see "Segmentation").

PERFORM Statement

Chart   2.   Logical Flow of Option 4 PERFORM Statement Varying One
             Identifier

```
                    ****A2*********
                    *             *
                    * EXECUTION OF*
                    * PERFORM STMT*
                    *    BEGINS   *
                    ***************



                    *****B2*********
                    *             *
                    *     SET     *
                    * IDENTIFIER-1*
                    * EQUAL TO ITS*
                    *  FROM VALUE *
                    ***************



                      C2 *.                   ****C3*********
                     *      *.                 *             *
                    *  TEST   *.  TRUE         *             *
              >*.   CONDITION-1 .*-------->*    EXIT       *
                    *.       .*             *             *
                     *.    .*                ***************
                       *. .*
                        *FALSE



                    ****D2*********
                    *             *
                    *   EXECUTE   *
                    * PROCEDURE-1 *
                    *    THRU     *
                    * PROCEDURE-2 *
                    ***************



                    *****E2*********
                    *             *
                    *   AUGMENT   *
              ----* IDENTIFIER-1*
                    * WITH ITS BY *
                    *    VALUE    *
                    ***************
```
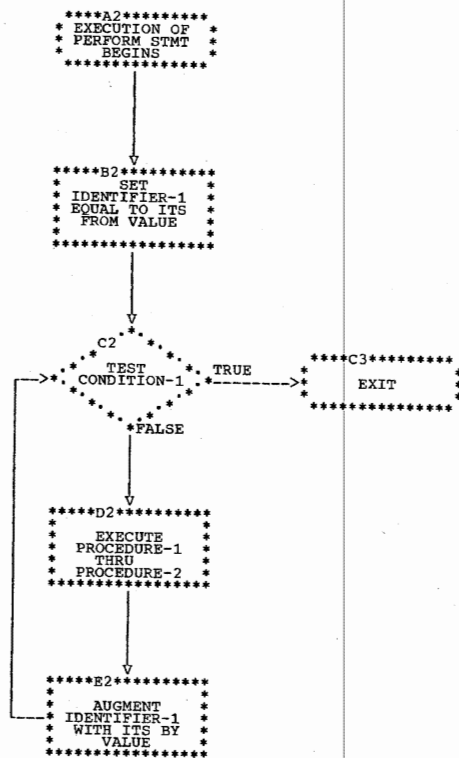
Chart   3.   Logical Flow of Option 4 PERFORM Statement Varying Two
             Identifiers

```
            ****A1*********
            * EXECUTION OF *
            * PERFORM STMT *
            *    BEGINS    *
            ***************
                   |
                   v
            *****B1*********
            *             *
            * IDENTIFIER-1 *
            * IDENTIFIER-4 *
            *SET TO INITIAL*
            *  FROM VALUE  *
            ***************
            ****
            *  *
            * C1 *->
            *  *
            ****    |
                   v
                .*.
              C1  *.                     ****C2*********
            *       *.  TRUE            *             *
          *   TEST    *.-------------->*     EXIT      *
          *.CONDITION-1.*               *             *
            *.       .*                 ***************
              *. .*
                *FALSE
                   |
                   v
                .*.
              D1  *.
            *       *.  TRUE
      ->*.   TEST    *.------------------------
          *.CONDITION-2.*                       |
            *.       .*                          |
              *. .*                              |
                *FALSE                           |
                   |                             |
                   v                             v
            *****E1*********              *****E2*********
            *             *              *     SET       *
            *  EXECUTE     *              *IDENTIFIER-4 TO*
            * PROCEDURE-1  *              *  ITS INITIAL  *
            *   THRU       *              *  FROM VALUE   *
            * PROCEDURE-2  *              *             *
            ***************              ***************
                   |                             |
                   v                             v
            *****F1*********              *****F2*********
            *  AUGMENT     *              *   AUGMENT     *
      ---*  IDENTIFIER-4 *              * IDENTIFIER-1  *
            * WITH ITS BY  *              * WITH ITS BY   *
            *   VALUE      *              *    VALUE      *
            ***************              ***************
                                                 |
                                                 v
                                                ****
                                                *  *
                                                * C1 *
                                                *  *
                                                ****
```
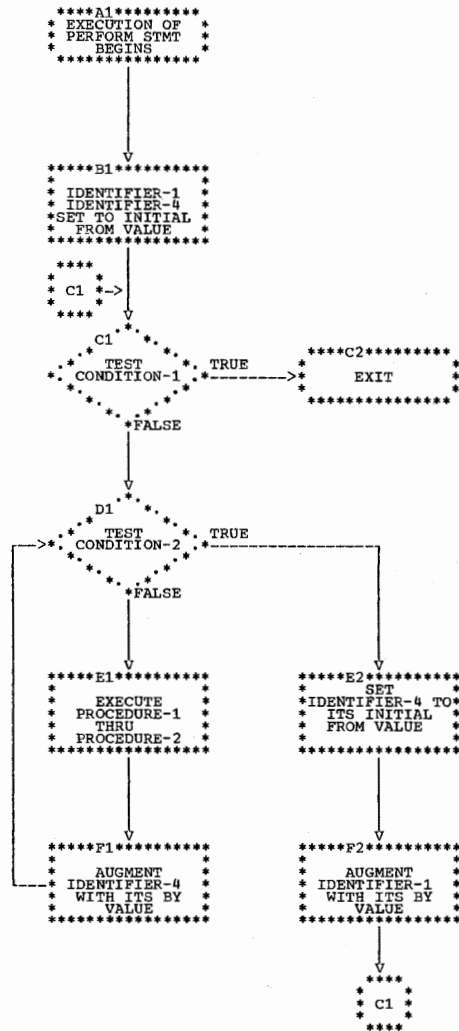
Chart   4.   Logical Flow of Option 4 PERFORM Statement Varying Three
             Identifiers

```
                              ****A2*********
                              * EXECUTION OF *
                              * PERFORM STMT *
                              *    BEGINS    *
                              ***************
                                     |
                                     v
                              *****B2*********
                              *  IDENTIFIER-1 *
                              *  IDENTIFIER-4 *
                              *  IDENTIFIER-7 *
                              *SET TO INITIAL *
                              *  FROM VALUES  *
                              ******************
                                     |
                                     v
        ****               C2  *  *.                  ****C3*********
      *      *           .*       *.   .TRUE          *             *
      *  C2  *----->*. CONDITION-1 .*------->*        EXIT          *
      *      *        *.  TEST   .*                   ***************
        ****            *.     .*
                          *. .*
                          *FALSE
                            |
                            v
        ****               D2  *  *.                                                  TRUE
      *      *           .*       *.   .TRUE
      *  D2  *----->*. CONDITION-2 .*-------------------------------------------------
      *      *        *.  TEST   .*
        ****            *.     .*
                          *. .*
                          *FALSE
                            |
                            v
                           E2  *  *.
                       ->.*       *.   .TRUE
                         *. CONDITION-3 .*-----------------
                          *.  TEST   .*
                            *.     .*
                              *. .*
                              *FALSE
                                |
                                v
        *****F2*********          *****F3*********          *****F4*********
        *             *          *     SET      *          *     SET      *
        *   EXECUTE   *          * IDENTIFIER-7 *          *IDENTIFIER-4 TO*
        * PROCEDURE-1 *          *TO ITS INITIAL*          *  ITS INITIAL *
        *    THRU     *          *  FROM VALUE  *          *  FROM VALUE  *
        * PROCEDURE-2 *          ***************          ***************
        ***************                 |                        |
               |                        v                        v
               v                *****G3*********          *****G4*********
        *****G2*********        *   AUGMENT    *          *   AUGMENT    *
        *   AUGMENT    *        * IDENTIFIER-4 *          * IDENTIFIER-1 *
        * IDENTIFIER-7 *        * WITH ITS BY  *          * WITH ITS BY  *
        * WITH ITS BY  *        *    VALUE     *          *    VALUE     *
        *    VALUE     *        ***************          ***************
        ***************               |                        |
                                      v                        v
                                    ****                     ****
                                  *      *                 *      *
                                  *  D2  *                 *  C2  *
                                  *      *                 *      *
                                    ****                     ****
```

STOP Statement

The STOP statement halts the object program either permanently or temporarily.

```
r-----------------------------------------------------------------------------
|                                   Format                                    |
|----------------------------------------------------------------------------|
|                                                                            |
|                  (  RUN      )                                             |
|       STOP      {            }                                            |
|                  (  literal  )                                            |
|                                                                            |
L-----------------------------------------------------------------------------
```

When the RUN option is used, the execution of the object program is terminated, and control is returned to the system.

If a STOP statement with the RUN option appears in an imperative statement, it must appear as the only or last statement in a sequence of imperative statements. All files should be closed before a STOP RUN statement is issued.

For the effect when STOP RUN is used in either a calling program or a called program, see "Subprogram Linkage."

When the literal option is used, the literal is communicated to the operator. The program may be resumed only by operator intervention. Continuation of the object program begins with the execution of the next statement in sequence.

The literal may be numeric or nonnumeric, or it may be any figurative constant except ALL.

EXIT Statement

The EXIT statement provides a common end point for a series of procedures.

```
r-----------------------------------------------------------------------------
|                                   Format                                    |
|----------------------------------------------------------------------------|
|                                                                            |
|      paragraph-name.   EXIT [PROGRAM].                                     |
|                                                                            |
L-----------------------------------------------------------------------------
```

It is sometimes necessary to transfer control to the end point of a series of procedures. This is normally done by transferring control to the next paragraph or section, but in some cases this does not have the required effect. For instance, the point to which control is to be transferred may be at the end of a range of procedures governed by a

EXIT Statement

PERFORM or at the end of a declarative section. The EXIT statement is
provided to enable a procedure-name to be associated with such a point.

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

    The EXIT statement must be preceded by a paragraph-name and be the
only statement in the paragraph.

    The EXIT statement with the PROGRAM option is discussed in
"Subprogram Linkage."

Movement and inspection of data are implicit in the functioning of several of the COBOL statements. These statements are: MOVE, EXAMINE, and TRANSFORM.

MOVE Statement

The MOVE statement is used to transfer data from one area of storage to one or more other areas.

```
---------------------------------------------------------------------
|                            Format 1                               |
|-------------------------------------------------------------------|
|                                                                   |
|         ( identifier-1 )                                          |
|   MOVE  {             }  TO  identifier-2  [identifier-3]...       |
|         ( literal     )                                           |
|                                                                   |
---------------------------------------------------------------------
```

```
---------------------------------------------------------------------
|                            Format 2                               |
|-------------------------------------------------------------------|
|                                                                   |
|         ( CORRESPONDING )                                         |
|   MOVE  {              }  identifier-1 TO identifier-2            |
|         ( CORR         )                                          |
|                                                                   |
---------------------------------------------------------------------
```

In Format 1, identifier-1 and literal represent the sending area; identifier-2, identifier-3, ... represent the receiving areas.

The data designated by literal or identifier-1 is moved first to identifier-2, then to identifier-3 (if specified), etc.

An index data item cannot appear as an operand of a MOVE statement.

In Format 2, the CORRESPONDING option is used to transfer data between items of the same name simply by specifying the group items to which they belong.

Neither identifier may be a level-66, level-77, or level-88 data item.

Data items from each group are considered CORRESPONDING when they have the same name and qualification, up to but not including identifier-1 and identifier-2.

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

Each elementary item containing an OCCURS, REDEFINES, USAGE IS INDEX, or RENAMES clause is ignored. However, either identifier may have a

REDEFINES or OCCURS clause in its description or may be subordinate to a data item described with these clauses.

General Rules Applying to Any MOVE Statement:

1. Any move in which the sending and receiving items are both elementary items is an elementary move. Each elementary item belongs to one of the following categories: numeric, alphabetic, alphanumeric, numeric edited, or alphanumeric edited (see "PICTURE Clause" in "Data Division"). Numeric literals belong to the category numeric; nonnumeric literals belong to the category alphanumeric.

2. When an alphanumeric edited, alphanumeric, or alphabetic item is a receiving item:

    a. Justification and any necessary filling of unused character positions takes place as defined under the JUSTIFIED clause. Unused character positions are filled with spaces.

    b. If the size of the sending item is greater than the size of the receiving item, the excess characters are truncated after the receiving item is filled.

    c. If the sending item has an operational sign, the absolute value is used.

3. When a numeric or numeric edited item is a receiving item:

    a. Alignment by decimal point and any necessary zero filling of unused character positions takes place, except when zeros are replaced because of editing requirements.

    b. The absolute value of the sending item is used if the receiving item has no operational sign.

    c. If the sending item has more digits to the left or right of the decimal point than the receiving item can contain, excess digits are truncated.

    d. The results at object time may be unpredictable if the sending item contains any nonnumeric characters.

4. Any necessary conversion of data from one form of internal representation to another takes place during the move, along with any specified editing in the receiving item.

5. Any move that is not an elementary move is treated exactly as though it were an alphanumeric elementary move, except that there is no conversion of data from one form of internal representation to another.

6. When the sending and receiving operands of a MOVE statement share a part of their storage (that is, when the operands overlap), the result of the execution of such a statement is unpredictable.

There are certain restrictions on elementary moves. These are shown in Table 15.

Table 15. Permissible Moves

Source Field \ Receiving Field	GR	AL	AN	ED	BI	NE	ANE	ID	EF	IF	SN	SR
Group (GR)	Y	Y	Y	Y¹	Y¹	Y¹	Y¹	Y¹	Y¹	Y¹	Y¹	Y¹
Alphabetic (AL)	Y	Y	Y	N	N	N	Y	N	N	N	N	N
Alphanumeric (AN)	Y	Y	Y	Y⁴	Y⁴	Y⁴	Y	Y⁴	Y⁴	Y⁴	Y⁴	Y⁴
External Decimal (ED)	Y¹	N	Y²	Y	Y	Y	Y²	Y	Y	Y	Y	Y
Binary (BI)	Y¹	N	Y²	Y	Y	Y	Y²	Y	Y	Y	Y	Y
Numeric Edited (NE)	Y	N	Y	N	N	N	Y	N	N	N	N	N
Alphanumeric Edited (ANE)	Y	Y	Y	N	N	N	Y	N	N	N	N	N
ZEROS (numeric or alphanumeric)	Y	N	Y	Y³	Y³	Y³	Y	Y³	Y³	Y³	Y³	Y³
SPACES (AL)	Y	Y	Y	N	N	N	Y	N	N	N	N	N
HIGH-VALUE, LOW-VALUE, QUOTES	Y	N	Y	N	N	N	Y	N	N	N	N	N
ALL literal	Y	Y	Y	Y⁵	Y⁵	Y⁵	Y	Y⁵	N	N	N	N
Numeric Literal	Y¹	N	Y²	Y	Y	Y	Y²	Y	Y	Y	Y	Y
Nonnumeric Literal	Y	Y	Y	Y⁵	Y⁵	Y⁵	Y	Y⁵	N	N	N	N
Internal Decimal (ID)	Y¹	N	Y²	Y	Y	Y	Y²	Y	Y	Y	Y	Y
External Floating-point (EF)	Y¹	N	N	Y	Y	Y	N	Y	Y	Y	Y	Y
Internal Floating-point (IF)	Y¹	N	N	Y	Y	Y	N	Y	Y	Y	Y	Y
Sterling Nonreport (SN)	Y¹	N	Y	Y	Y	Y	N	Y	Y	Y	Y	Y
Sterling Report (SR)	Y	N	Y	N	N	N	Y	N	N	N	N	N
Floating-point Literal	Y¹	N	N	Y	Y	Y	N	Y	Y	Y	Y	Y

¹Move without conversion (like AN to AN)
²Only if the decimal point is at the right of the least significant digit
³Numeric move
⁴The alphanumeric field is treated as an ED (integer) field
⁵The literal must consist only of numeric characters and is treated as an ED integer field

EXAMINE Statement

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

```
+------------------------------------------------------------------+
|                           Format 1                               |
+------------------------------------------------------------------+
|                                                                  |
|                                    ( UNTIL FIRST )               |
|    EXAMINE identifier TALLYING     { ALL         }  literal-1    |
|                                    ( LEADING     )               |
|                                                                  |
|          [REPLACING BY literal-2]                                |
|                                                                  |
+------------------------------------------------------------------+
```

```
+------------------------------------------------------------------+
|                           Format 2                               |
+------------------------------------------------------------------+
|                                    ( ALL         )               |
|                                    { LEADING     }               |
|    EXAMINE identifier REPLACING    { FIRST       }  literal-1    |
|                                    ( UNTIL FIRST )               |
|                                                                  |
|          BY   literal-2                                          |
|                                                                  |
+------------------------------------------------------------------+
```

In all cases, the description of identifier must be such that its usage is display (explicitly or implicitly).

When identifier represents a nonnumeric data item, examination starts at the leftmost character and proceeds to the right. Each character in the data item is examined in turn.

When identifier represents a numeric data item, this data item must consist of numeric characters, and may possess an operational sign. Examination starts at the leftmost character and proceeds to the right. Each character is examined in turn.

If the letter 'S' is used in the PICTURE of the data item description to indicate the presence of an operational sign, the sign is ignored by the EXAMINE statement.

Each literal must consist of a single character belonging to a class consistent with that of the identifier; in addition, each literal may be any figurative constant except ALL. If identifier is numeric, each literal must be an unsigned integer or the figurative constant ZERO (ZEROES, ZEROS).

When Format 1 is used, an integral count is created which replaces the value of a special register called TALLY, whose implicit description is that of an unsigned integer of five digits (see "Language Considerations").

1. When the ALL option is used, this count represents the number of occurrences of literal-1.

2. When the LEADING option is used, this count represents the number of occurrences of <u>literal-1</u> prior to encountering a character other than literal-1.

3. When the UNTIL FIRST option is used, this count represents all characters encountered before the first occurrence of <u>literal-1</u>.

Whether Format 2 is used, or the REPLACING option of Format 1, the replacement rules are the same. They are as follows:

1. When the ALL option is used, <u>literal-2</u> is substituted for each occurrence of <u>literal-1</u>.

2. When the LEADING option is used, the substitution of <u>literal-2</u> for each occurrence of <u>literal-1</u> terminates as soon as a character other than literal-1 or the right-hand boundary of the data item is encountered.

3. When the UNTIL FIRST option is used, the substitution of <u>literal-2</u> terminates as soon as <u>literal-1</u> or the right-hand boundary of the data item is encountered.

4. When the FIRST option is used, the first occurrence of <u>literal-1</u> is replaced by <u>literal-2</u>.

Specific EXAMINE statements showing the effect of each statement on the associated data item and the TALLY are shown in Table 16.

Table 16. Examples of Data Examination

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

TRANSFORM Statement

The TRANSFORM statement is used to alter characters according to a transformation rule.

```
+----------------------------------------------------------------------+
|                              Format                                   |
+----------------------------------------------------------------------+
|                                                                      |
|                                          ( figurative-constant-1 )   |
|    TRANSFORM identifier-3 CHARACTERS FROM { nonnumeric-literal-1  }   |
|                                          ( identifier-1           )   |
|                                                                      |
|           ( figurative-constant-2 )                                  |
|        TO { nonnumeric-literal-2  }                                  |
|           ( identifier-2          )                                  |
|                                                                      |
+----------------------------------------------------------------------+
```

Identifier-3 must represent an elementary alphabetic, alphanumeric, or numeric edited item, or a group item.

The combination of the FROM and TO options determines what the transformation rule is.

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

1. Nonnumeric literals require enclosing quotation marks.

2. Identifier-1 and identifier-2 must be elementary alphabetic, or alphanumeric items, or fixed length group items not over 255 characters in length.

3. A character may not be repeated in nonnumeric-literal-1 or in the area defined by identifier-1. If a character is repeated, the results will be unpredictable.

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

When either identifier-1 or identifier-2 appears as an operand of the specific transformation, the user can change the transformation rule at object time.

Examples of data transformation are given in Table 17; combinations of the FROM and TO options are shown in Table 18.

If any of the operands of a TRANSFORM statement share a part of their storage (that is, if the operands overlap), the result of the execution of such a statement is unpredictable.

Table 17. Examples of Data Transformation

Identifier-3 (Before)	FROM	TO	Identifier-3 (After)
1b7bbABC	SPACE	QUOTE	1"7""ABC
1b7bbABC	"17CB"	"QRST"	QbRbbATS
1b7bbABC	b17ABC	CBA71b	BCACC71b
1234WXY89	98YXW4321	ABCDEFGHI	IHGFEDCBA

Table 18. Combinations of FROM and TO Options (Part 1 of 2)

Operands	Transformation Rule
FROM figurative-constant-1 TO figurative-constant-2	All characters in the data item represented by identifier-3 equal to the single character figurative-constant-1 are replaced by the single character figurative-constant-2.
FROM figurative-constant-1 TO nonnumeric-literal-2	All characters in the data item represented by identifier-3 equal to the single character figurative-constant-1 are replaced by the single character nonnumeric-literal-2.
FROM figurative-constant-1 TO identifier-2	All characters in the data item represented by identifier-3 equal to the single character figurative-constant-1 are replaced by the single character represented by identifier-2.
FROM nonnumeric-literal-1 TO figurative-constant-2	All characters in the data item represented by identifier-3 that are equal to any character in nonnumeric-literal-1 are replaced by the single character figurative-constant-2.
FROM nonnumeric-literal-1 TO nonnumeric-literal-2	Nonnumeric-literal-1 and nonnumeric-literal-2 must be equal in length or nonnumeric-literal-2 must be a single character. If the nonnumeric-literals are equal in length, any character in the data item represented by identifier-3 equal to a character in nonnumeric-literal-1 is replaced by the character in the corresponding position of nonnumeric-literal-2. If the length of nonnumeric-literal-2 is one, all characters in the data item represented by identifier-3 that are equal to any character appearing in nonnumeric-literal-1 are replaced by the single character given in nonnumeric-literal-2.

TRANSFORM Statement

Table 18. Combinations of FROM and TO Options (Part 2 of 2)

Operands	Transformation Rule
FROM nonnumeric-literal-1 TO identifier-2	Nonnumeric-literal-1 and the data item represented by identifier-2 must be equal in length or identifier-2 must represent a single character item. If nonnumeric-literal-1 and identifier-2 are equal in length, any character represented by identifier-3 equal to a character in nonnumeric-literal-1 is replaced by the character in the corresponding position of the item represented by identifier-2. If the length of the data item represented by identifier-2 is one, all characters represented by identifier-3 that are equal to any character appearing in nonnumeric-literal-1 are replaced by the single character represented by identifier-2.
FROM identifier-1 TO figurative-constant-2	All characters represented by identifier-3 that are equal to any character in the data item represented by identifier-1 are replaced by the single character figurative-constant-2.
FROM identifier-1 TO nonnumeric-literal-2	The data item represented by identifier-1 and nonnumeric-literal-2 must be of equal length or nonnumeric-literal-2 must be one character. If identifier-1 and nonnumeric-literal-2 are equal in length, any character in identifier-3 equal to a character in identifier-1 is replaced by the character in the corresponding position of nonnumeric-literal-2. If the length of nonnumeric-literal-3 is one, all characters represented by identifier-3 that are equal to any character represented by identifier-1 are replaced by the single character given in nonnumeric-literal-2.
FROM identifier-1 TO identifier-2	Any character in the data item represented by identifier-3 equal to a character in the data item represented by identifier-1 is replaced by the character in the corresponding position of the data item represented by identifier-2. Identifier-1 and identifier-2 can be one or more characters, but must be equal in length.

The flow of data through the computer is governed by the Disk Operating System. The COBOL statements discussed in this section are used to initiate the flow of data to and from files stored on external media and to govern low-volume information that is to be obtained from or sent to input/output devices such as a card reader or console typewriter.

The Disk Operating System is a record processing system. That is, the unit of data made available by a READ or passed along by a WRITE is the record. The COBOL user need be concerned only with the use of individual records; provision is automatically made for such operations as the movement of data into buffers and/or internal storage, validity checking, error correction (where feasible), unblocking and blocking, and volume switching procedures.

Discussions in this section use the terms volume and reel. The term volume applies to all input/output devices. The term reel applies only to tape devices. Treatment of mass storage devices in the sequential access mode is logically equivalent to the treatment of tape files.

Note: The WRITE statement with the BEFORE/AFTER ADVANCING option is referred to in some of the discussions that follow as the WRITE BEFORE/AFTER ADVANCING statement. Similarly, the WRITE statement with the AFTER POSITIONING option is referred to in some discussions as the WRITE AFTER POSITIONING statement.

OPEN Statement

The OPEN statement initiates the processing of input, output, and input-output files. It performs checking and/or writing of labels and other input/output operations.

```
┌─────────────────────────────────────────────────────────────────────┐
│                              Format                                   │
├─────────────────────────────────────────────────────────────────────┤
│                                                                       │
│                             ┌ REVERSED         ┐                      │
│     OPEN [INPUT {file-name  │                  │  }...]               │
│                             └ WITH NO REWIND    ┘                     │
│                                                                       │
│         [OUTPUT {file-name [WITH NO REWIND] }...]                     │
│                                                                       │
│         [I-O  {file-name} ...]                                        │
│                                                                       │
└─────────────────────────────────────────────────────────────────────┘
```

At least one of the options INPUT, OUTPUT, or I-O must be specified. However, there must be no more than one instance of each option in the same statement, although more than one file-name may be used with each option. These options may appear in any order.

OPEN Statement

The I-O option pertains only to mass storage files.

The file-name must be defined by a file description entry in the Data
Division.

The OPEN statement must not specify a sort-file; an OPEN statement
must be specified for all other files. The OPEN statement for a file
must be executed prior to the first READ, SEEK, START, or WRITE
statement for that file. A second OPEN statement for a file cannot be
executed prior to the execution of a CLOSE statement for that file. The
OPEN statement does not obtain or release the first data record. A READ
or WRITE statement must be executed to obtain or release, respectively,
the first data record.

When checking or writing the first label, the OPEN statement causes
the user's beginning label subroutine to be executed if one is specified
by a USE sentence in the Declaratives.

For an indexed output file, execution of the OPEN statement causes
the user's error declarative to be executed (if one was specified) when
the space allocated for the cylinder index or master index is not large
enough to contain the number of entries required for the prime data.

The REVERSED and the NO REWIND options can be used only with
sequential single reel files. The REVERSED option may be specified only
for a file containing fixed-length (F mode) records.

Files with nonstandard header labels must not be opened for reversed
reading unless the last header label is followed by a tape mark.

For tape files, the following rules apply:

1. When neither the REVERSED nor the NO REWIND option is specified,
    execution of the OPEN statement causes the file to be positioned at
    its beginning.

2. When either the REVERSED or the NO REWIND option is specified,
    execution of the OPEN statement does not cause the file to be
    repositioned. When the REVERSED option is specified, the file must
    have been previously positioned at its end. When the NO REWIND
    option is specified, the file must have been previously positioned
    at its beginning.

When the REVERSED option is specified, subsequent READ statements for
the file make the data records of the file available in reversed order;
that is, starting with the last record.

If an input file is designated with the OPTIONAL clause in the File
Control paragraph of the Environment Division (sequential file
processing), the clause is treated as comments. The desired effect is
achieved by specifying the IGN parameter in the ASSGN control statement
for the file. If the file is not present, the first READ statement for
this file causes control to be passed to the imperative statement in the
AT END phrase.

The I-O option permits the opening of a mass storage file for both
input and output operations. Since this option implies the existence of

the file, it cannot be used if the mass storage file is being initially created.

When the I-O option is used, the execution of the OPEN statement includes the following steps:

1. The label is checked.

2. The user's label subroutine, if one is specified by a USE sentence, is executed.

3. The label is written.

A file may be opened as INPUT and OUTPUT and I-O in any order (with intervening CLOSE statements without the UNIT or REEL option). However, an indexed file may not be created and read or updated within the same program.

START Statement

The START statement initiates processing of a segment of a sequentially accessed indexed file at a specified key.

```
--------------------------------------------------------------------
|                               Format                             |
|------------------------------------------------------------------|
|                                                                  |
|    START file-name INVALID KEY imperative-statement              |
|                                                                  |
--------------------------------------------------------------------
```

The file-name must be defined by a file description entry in the Data Division.

The key at which processing is to begin must be stored in the data-name specified in the NOMINAL KEY clause before executing the START statement.

Normally, an indexed file in the sequential mode is processed sequentially from the first record to the last or until the file is closed. If processing is to begin at other than the first record, a START statement must be executed after the OPEN but before the first READ statement. Processing will then continue sequentially until a START statement or a CLOSE statement is executed or until the end-of-file is reached.

If processing is to begin at the first record, a START statement is not required before the first READ.

Control is passed to the imperative-statement following INVALID KEY when the contents of the NOMINAL KEY field are invalid. The key is considered invalid when the record is not found in the file.

SEEK Statement

The SEEK statement is meant to initiate the accessing of a mass storage data record for subsequent reading or writing. It is used to optimize programming efficiency.

```
r----------------------------------------------------------------------------1
|                                   Format                                    |
|----------------------------------------------------------------------------|
|                                                                            |
|    SEEK file-name RECORD                                                    |
|                                                                            |
L----------------------------------------------------------------------------J
```
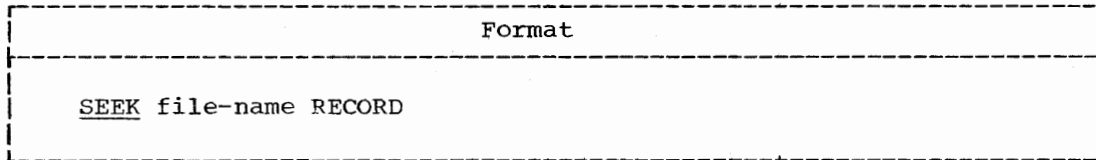
A SEEK statement pertains only to direct files in the random access mode and may be executed prior to the execution of a READ or WRITE statement.

The file-name must be defined by a file description entry in the Data Division.

The SEEK statement uses the contents of the data-name in the ACTUAL KEY clause for the location of the record to be accessed. At the time of execution, the determination is made as to the validity of the contents of the ACTUAL KEY data item for the particular mass storage file. If the key is invalid, when the next READ or WRITE statement for the associated file is executed, control will be given to the imperative-statement in the INVALID KEY option.

Two SEEK statements for the same direct file may logically follow each other. Any validity check associated with the first SEEK statement is negated by the execution of the second SEEK statement.

If the contents of the ACTUAL KEY are altered between the SEEK statement and the subsequent READ or WRITE statement, any validity check associated with the SEEK statement is negated, and the READ or WRITE statement is processed as if no SEEK statement preceded it.

READ Statement

The functions of the READ statement are:

1. For sequential file processing, to make available the next logical record from an input file and to give control to a specified imperative-statement when end-of-file is detected.

2. For random file processing, to make available a specific record from a mass storage file and to give control to a specified imperative-statement if the contents of the associated ACTUAL KEY or NOMINAL KEY data item are found to be invalid.

```
r--------------------------------------------------------------------¬
|                              Format                                |
+--------------------------------------------------------------------+
|                                                                    |
|    READ file-name  RECORD [INTO identifier]                        |
|                                                                    |
|      ( AT END      )                                               |
|      {            }  imperative-statement                          |
|      ( INVALID KEY )                                               |
|                                                                    |
L--------------------------------------------------------------------
```

An OPEN statement must be executed for the file prior to the execution of the first READ for that file.

When a READ statement is executed, the next logical record in the named file becomes accessible in the input area defined by the associated record description entry.

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

When a file consists of more than one type of logical record, these records automatically share the same storage area; this is equivalent to an implicit redefinition of the area. Only the information that is present in the current record is accessible.

FILE-NAME: The file-name must be defined by a file description entry in the Data Division.

INTO IDENTIFIER OPTION: The INTO identifier option makes the READ statement equivalent to a READ statement plus a MOVE statement. Identifier must be the name of a Working-Storage or Linkage Section entry, or an output record of a previously opened file. When this option is used, the current record becomes available in the input area, as well as in the area specified by identifier. Data will be moved into identifier in accordance with the COBOL rules for the MOVE statement without the CORRESPONDING option.

AT END OPTION: The AT END option must be specified for all files in the sequential access mode. If, during the execution of a READ statement, the logical end of the file is reached, control is passed to the imperative-statement specified in the AT END phrase. After execution of the imperative-statement associated with the AT END phrase, a READ statement for that file must not be given without prior execution of a CLOSE statement and an OPEN statement for that file.

READ Statement

If an input file is designated with the OPTIONAL clause in the File Control paragraph of the Environment Division (sequential file processing), the clause is treated as comments. The desired effect is achieved by specifying the IGN parameter in the ASSGN control statement for the file. If the file is not present, the first READ statement for this file causes the imperative-statement in the AT END phrase to be executed.

If, during the processing of a multivolume file in the sequential access mode, end-of-volume is recognized on a READ, the following operations are carried out:

1. The standard ending volume label procedure and the user's ending volume label procedure, if specified by the USE sentence. The order of execution of these two procedures is specified by the USE sentence.

2. A volume switch.

3. The standard beginning volume label procedure and the user's beginning volume label procedure, if specified. The order of execution is again specified by the USE sentence.

4. The first data record of the new volume is made available.

INVALID KEY OPTION: If ACCESS IS RANDOM is specified for the file, the contents of the ACTUAL or NOMINAL KEY for the file must be set to the desired value before the execution of the READ statement.

Only the track specified in the ACTUAL KEY is searched for the record being read.

When APPLY EXTENDED-SEARCH is used, the entire cylinder is searched if the desired record cannot be found on the specified track.

For a randomly accessed file, the READ statement implicitly performs the functions of the SEEK statement, unless a SEEK statement for the file has been executed prior to the READ statement.

The INVALID KEY option must be specified for files in the random access mode. The imperative-statement following INVALID KEY is executed when the contents of the ACTUAL KEY or NOMINAL KEY field are invalid.

The key is considered invalid under the following conditions:

1. For a direct file that is accessed randomly, when the record is not found within the search limits, or when the track address in the ACTUAL KEY field is outside the limits of the file.

2. For an indexed file that is accessed randomly, when no record exists whose RECORD KEY field matches the contents of the NOMINAL KEY field.

WRITE Statement

The WRITE statement releases a logical record to an output file. It can also be used for vertical positioning of a print file. For sequentially accessed mass storage files, the WRITE statement passes control to a specified imperative-statement if the file limit is exceeded. For randomly accessed mass storage files, the WRITE statement passes control to a specified imperative-statement if the contents of the associated ACTUAL or NOMINAL KEY data item are found to be invalid.

The WRITE statement can also be used for pocket selection for a card punch file.

```
+--------------------------------------------------------------------+
|                             Format 1                               |
+--------------------------------------------------------------------+
|                                                                    |
|   WRITE record-name [FROM identifier-1]                            |
|                                                                    |
|       ⎡⎧BEFORE⎫                     ⎧identifier-2 LINES⎫⎤          |
|       ⎢⎨      ⎬  ADVANCING          ⎨integer LINES     ⎬⎥          |
|       ⎣⎩AFTER ⎭                     ⎩mnemonic-name      ⎭⎦          |
|                                                                    |
|        ┌─────────────────────────────────────────────────┐        |
|        │      ⎧END-OF-PAGE⎫                               │        |
|        │[AT   ⎨           ⎬     imperative-statement]     │        |
|        │      ⎩EOP        ⎭                               │        |
|        └─────────────────────────────────────────────────┘        |
|                                                                    |
+--------------------------------------------------------------------+
```

```
+--------------------------------------------------------------------+
|                             Format 2                               |
+--------------------------------------------------------------------+
|                                                                    |
|   WRITE record-name [FROM identifier-1]                            |
|                                                                    |
|                               ⎧identifier-2⎫                       |
|       AFTER POSITIONING        ⎨            ⎬     LINES            |
|                               ⎩integer     ⎭                       |
|                                                                    |
|            ⎧END-OF-PAGE⎫                                           |
|       [AT  ⎨           ⎬     imperative-statement]                |
|            ⎩EOP        ⎭                                           |
|                                                                    |
+--------------------------------------------------------------------+
```

```
+--------------------------------------------------------------------+
|                             Format 3                               |
+--------------------------------------------------------------------+
|                                                                    |
|   WRITE record-name [FROM identifier-1]                            |
|                                                                    |
|       INVALID KEY imperative-statement                            |
|                                                                    |
+--------------------------------------------------------------------+
```

An OPEN statement for a file must be executed prior to executing the first WRITE statement for that file.

For files in both the sequential and random access modes, the logical record released is no longer available after the WRITE statement is executed.

RECORD-NAME: The record-name is the name of a logical record in the File Section of the Data Division and must not be part of a sort-file.

FROM OPTION: When the FROM option is written, the WRITE statement is equivalent to the statement MOVE identifier-1 TO record-name followed by the statement WRITE record-name. Data is moved into record-name in accordance with the COBOL rules for the MOVE statement without the CORRESPONDING option. Identifier-1 should be defined in the Working-Storage Section, the Linkage Section, or in another FD.

WRITE Statement

FORMAT 1 AND FORMAT 2:   Formats 1 and 2 are used only with standard
sequential files.

The ADVANCING and POSITIONING options allow control of the vertical
positioning of each record on the printed page. If the ADVANCING or
POSITIONING option is not used, automatic advancing is provided to cause
single spacing. If the ADVANCING or POSITIONING option is used, auto-
matic advancing is overridden.

When the ADVANCING or POSITIONING option is written for a record in a
file, every WRITE statement for records in the same file must also
contain one of these options. The POSITIONING and ADVANCING options may
not both be specified for a file.

When the ADVANCING or POSITIONING option is used, the first character
in each logical record for the file must be reserved by the user for the
control character. The compiler will generate instructions to insert
the appropriate carriage control character as the first character in the
record. If the records are to be punched, the first character is used
for pocket selection. It is the user's responsibility to see that the
appropriate channels are punched on the carriage control tape.

Format 1:   In the ADVANCING option, when identifier-2 is used, it must
be the name of a nonnegative numeric elementary item (less than 100)
described as an integer. If identifier-2 is specified, the printer page
is advanced the number of lines contained in the identifier.

When integer is used in the ADVANCING option, it must be nonnegative,
and less than 100. If integer is specified, the printer page is
advanced the number of lines equal to the value of integer.

When the mnemonic-name option is used in the ADVANCING option, it
must be defined as a function-name in the Special-Names paragraph of the
Environment Division. It is used for a skip to channels 1-9, 10-12, and
to suppress spacing. It is also used for pocket selection for a card
punch file.

The meaning of each function-name is shown in Table 19.

Table 19.   Action Taken for Function-Names -- ADVANCING Option

Function-name	Action Taken
CSP	Suppress spacing
C01 through C09	Skip to channel 1 through 9, respectively
C10 through C12	Skip to channel 10, 11, and 12, respectively
S01, S02	Pocket select 1 or 2 on the IBM 1442, and P1 or P2 on the IBM 2540

If the BEFORE ADVANCING option is used, the record is written before
the printer page is advanced according to the preceding rules.

If the AFTER ADVANCING option is used, the record is written after
the printer page is advanced according to the preceding rules.

Format 2:   In the AFTER POSITIONING option, identifier-2 must be
described as a one-character alphanumeric item, that is, with PICTURE X.
Table 19 shows the valid values that identifier-2 may assume and their
interpretations.

In the AFTER POSITIONING option, integer must be unsigned, and must be the value 0, 1, 2, or 3. The values assume the meanings shown in Table 21.

Table 20. Values of Identifier-2 and Their Interpretation --
           POSITIONING Option

Value	Interpretation
b(blank)	Single-spacing
0	Double-spacing
-	Triple-spacing
+	Suppress spacing
1 - 9	Skip to channel 1 - 9, respectively
A, B, C	Skip to channel 10, 11, 12, respectively
V, W	Pocket select 1 or 2, respectively, on the IBM 1442, and P1 or P2 on the IBM 2540.

Table 21. Values of Integer and Their Interpretations -- POSITIONING
           Option

Value	Interpretation
0	Skip to channel 1 of next page (carriage control "eject")
1	Single-spacing
2	Double-spacing
3	Triple-spacing

If the AFTER POSITIONING option is used, the record is written after the printer page is advanced according to the preceding rules.

END-OF-PAGE Option: The END-OF-PAGE condition exists when the channel 12 punch on the carriage control tape is sensed by an on-line printer. The printer file must be defined as an unblocked single-buffered file. The programmer should ensure that every WRITE statement in the program (whether using the ADVANCING or the POSITIONING option) advances the printer only one line at a time; otherwise, the channel 12 punch may not be sensed and results may be unpredictable.

When an END-OF-PAGE condition exists, the writing and spacing operations are completed before the END-OF-PAGE imperative-statement is executed. The END-OF-PAGE statement will be executed only for an on-line printer.

Note: DISPLAY, EXHIBIT, WRITE AFTER POSITIONING, and WRITE AFTER ADVANCING statements all cause the printer to space before printing. However, a simple WRITE statement without any option given, or a WRITE BEFORE ADVANCING statement both cause the printer to space after printing. Therefore, it is possible that mixed DISPLAY statements, EXHIBIT statements, and simple WRITE statements or WRITE BEFORE ADVANCING statements within the same program may cause overprinting.

FORMAT 3: Format 3 is used for randomly or sequentially accessed mass storage files.

If ACCESS IS RANDOM is specified for the file, the contents of the ACTUAL or NOMINAL KEY field for the file must be set to the desired value before the execution of a WRITE statement. For a direct file, the track specified in the ACTUAL KEY field is searched for space for the record to be written.

The INVALID KEY phrase must be specified for a file that resides on a mass storage device. Control is passed to the imperative-statement following INVALID KEY when the following conditions exist:

1. For a mass storage file in the sequential access mode opened as OUTPUT, when an attempt is made to write beyond the limit of the file.

2. For a direct file opened as I-O or OUTPUT, if access is random and a record is being added to the file, when the track address specified in the ACTUAL KEY field is outside the limits of the file.

3. For a direct file opened as I-O, if access is random and a record is being updated, control is passed to the imperative-statement following INVALID KEY when the record is not found, or when the track number in the ACTUAL KEY field is outside the limits of the file.

4. For an indexed file opened as OUTPUT, if access is sequential, when either one of the following conditions occurs:

   a. The contents of the RECORD KEY field are not in ascending order when compared with the contents of the RECORD KEY field of the preceding record

   b. The contents of the RECORD KEY field duplicate those of the preceding record

5. For an indexed file opened as I-O, if access is random and a record is being added to the file, when the contents of the NOMINAL KEY field associated with the record to be added duplicate the contents of a RECORD KEY field already in the file.

RANDOMLY ACCESSED DIRECT FILES: The WRITE statement performs the function of a SEEK statement, unless a SEEK statement for this record was executed prior to the WRITE statement.

For a randomly accessed direct file that is opened I-O, the following considerations apply:

• If A or D is specified in the ASSIGN clause system-name, then:

   (1) a WRITE statement updates a record if the preceding input/output statement was a READ statement for a record with the same ACTUAL KEY.

   (2) a WRITE statement adds a new record to the file, whether or not a duplicate record exists, if the preceding READ statement was not for a record with the same ACTUAL KEY.

• If U or W is specified in the ASSIGN clause system-name, then:

   (1) a REWRITE statement searches for a record with a matching ACTUAL KEY, and updates it.

   (2) a WRITE statement adds a new record to the file, whether or not a duplicate key exists.

MULTIVOLUME SEQUENTIAL FILES: The discussion below applies to all multivolume tape files and to multivolume mass storage files in the sequential access mode.

   After the recognition of an end-of-volume on a multivolume OUTPUT or I-O file in the sequential access mode, the WRITE statement performs the following operations:

1.   The standard ending volume label procedure and the user's ending volume label procedure if specified by a USE sentence. The order of execution of these two procedures is specified by the USE sentence.

2.   A volume switch.

3.   The standard beginning volume label procedures and the user's beginning volume label procedure if specified by the USE sentence. The order is specified by the USE sentence.

REWRITE Statement

The function of the REWRITE statement is to replace a logical record on a mass storage device with a specified record, if the contents of the associated ACTUAL KEY or NOMINAL KEY are found to be valid.

```
┌─────────────────────────────────────────────────────────────────────────┐
│                                Format                                     │
├─────────────────────────────────────────────────────────────────────────┤
│                                                                           │
│   REWRITE record-name [FROM identifier]                                   │
│                                                                           │
│       INVALID KEY imperative-statement                                    │
│                                                                           │
└─────────────────────────────────────────────────────────────────────────┘
```

The READ statement for a file must be executed before a REWRITE statement for the file can be executed. A REWRITE statement can be executed only for direct or indexed files opened as I-O.

If ACCESS IS RANDOM is specified for the file, the ACTUAL or NOMINAL KEY must be set to the desired value prior to the execution of the REWRITE statement.

The record-name is the name of a logical record in the File Section of the Data Division, and must not be part of a sort file.

When the FROM option is used, the REWRITE statement is equivalent to the statement MOVE identifier TO record-name followed by the statement REWRITE record-name. Identifier should be defined in the Working-Storage Section, Linkage Section, or in another FD.

For a direct file that is accessed randomly, control is passed to the imperative-statement following INVALID KEY when the contents of the ACTUAL KEY field are invalid. The key is considered invalid when the record is not found, or when the track address is outside the limits of the file.

For an indexed file that is accessed randomly, control is passed to the imperative-statement following INVALID KEY when the preceding READ statement caused an error condition.

Note: For the relationship between the REWRITE statement and the ASSIGN clause system-name, see the paragraphs on Randomly Accessed Direct files in "WRITE Statement".

ACCEPT Statement

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

ACCEPT Statement

```
+---------------------------------------------------------------------+
|                              Format                                 |
+---------------------------------------------------------------------+
|                                                                     |
|                                ( SYSIPT      )                      |
|   ACCEPT identifier [FROM      { CONSOLE      }      ]              |
|                                ( mnemonic-name)                      |
|                                                                     |
+---------------------------------------------------------------------+
```

Identifier may be either a fixed-length group item or an elementary
alphabetic, alphanumeric, external decimal, or external floating-point
item. Identifier may not be any Special Register except TALLY. The
data is read and the appropriate number of characters is moved into the
area reserved for identifier. No editing or error checking of the
incoming data is done.

If the input/output device specified by an ACCEPT statement is the
same one designated for a READ statement, the results may be
unpredictable.

Mnemonic-name may assume either the meaning SYSIPT or CONSOLE.
Mnemonic-name must be specified in the Special-Names paragraph of the
Environment Division. If mnemonic-name is associated with CONSOLE,
identifier must not exceed 255 character positions in length. If the
FROM option is not specified, SYSIPT is assumed.

When an ACCEPT statement with the FROM mnemonic-name for CONSOLE
option or FROM CONSOLE is executed, the following actions are taken:

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

2.   Execution is suspended. When a console input message is identified
     by the control program, execution of the ACCEPT statement is
     resumed and the message is moved to the specified identifier and
     left justified regardless of the PICTURE. If the field is not
     filled the low-order positions may contain invalid data.

If mnemonic-name is associated with SYSIPT or the FROM SYSIPT option
is specified, an input record size of 80 is assumed. If the size of the
accepting data item is less than 80 characters, the data must appear as
the first set of characters within the input record. If the size of the
accepting data item is greater than 80 characters, as many input records
as necessary are read until the storage area allocated to the data item
is filled. If the accepting data item is greater than 80 characters,
but is not an exact multiple of 80, the remainder of the last input
record is not accessible.

DISPLAY Statement

The function of the DISPLAY statement is to write data on an output
device.
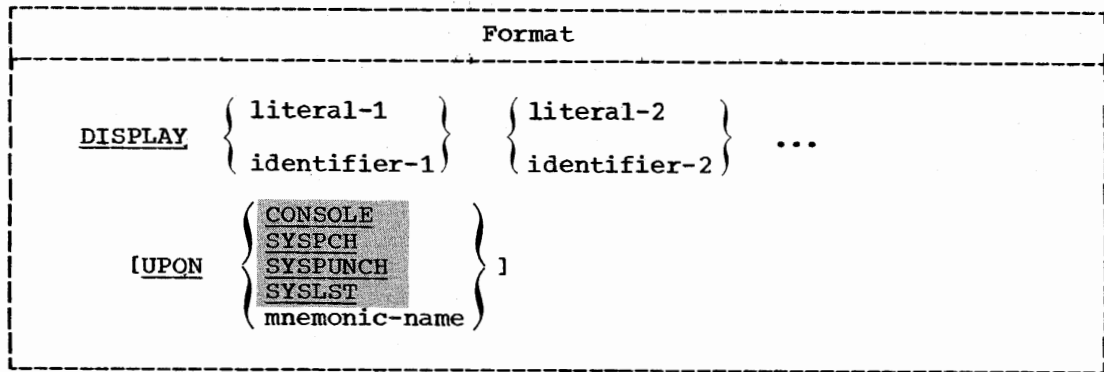
```
-------------------------------------------------------------------------
|                              Format                                     |
|-------------------------------------------------------------------------|
|                                                                         |
|   DISPLAY    ( literal-1    )   ( literal-2    )                         |
|              ( identifier-1 )   ( identifier-2 )   ...                   |
|                                                                         |
|              ( CONSOLE      )                                           |
|              ( SYSPCH       )                                           |
|   [UPON      ( SYSPUNCH     )  ]                                         |
|              ( SYSLST       )                                           |
|              ( mnemonic-name )                                          |
|                                                                         |
-------------------------------------------------------------------------
```

Mnemonic-name must be specified in the Special-Names paragraph of the Environment Division. Mnemonic-name may be associated only with the reserved words CONSOLE, SYSPCH, SYSPUNCH, or SYSLST.

When the UPON option is omitted, the system list device (SYSLST) is assumed.

A maximum logical record size is assumed for each device. For CONSOLE (the system logical console device), the maximum is 100 characters. For SYSLST (the system logical output device), the maximum is 120 characters. For SYSPCH or SYSPUNCH (the system punch device), the maximum is 72 characters, with positions 73-80 used for the PROGRAM-ID name.

If the total character count of all operands is less than the maximum (or 72 for SYSPCH or SYSPUNCH), the remaining character positions are padded with blanks. If the count exceeds the maximum size, operands are continued in the next record. As many records as necessary are written to display all the operands specified. Those operands pending at the time of the break are split between lines if necessary.

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

1. Internal-decimal and binary items are converted to external decimal. Signed values cause a low-order sign overpunch to be developed.

2. Internal floating-point items are converted to external floating-point.

3. No other data items require conversion.

For example, if three internal-decimal items have values of -34, +34, and 34, they are displayed as 3M, 3D, and 34, respectively.

If a figurative constant is specified as one of the operands, only a single occurrence of the figurative constant is displayed.

Identifier may not be any Special Register except TALLY.

When a DISPLAY statement contains more than one operand, the data contained in the first operand is stored as the first set of characters, and so on, until the output record is filled. This operation continues until all information is displayed. Data contained in an operand may extend into subsequent records.

Note: DISPLAY, EXHIBIT, WRITE AFTER POSITIONING, and WRITE AFTER
ADVANCING statements all cause the printer to space before printing.
However, a simple WRITE statement without any option given, or a WRITE
BEFORE ADVANCING statement both cause the printer to space after
printing.   Therefore, it is possible that mixed DISPLAY statements,
EXHIBIT statements, and simple WRITE statements or WRITE BEFORE
ADVANCING statements within the same program may cause overprinting.

CLOSE Statement

The CLOSE statement terminates the processing of input and output
reels, units, and files, with optional rewind and/or lock where
applicable.


```
+---------------------------------------------------------------------+
|                               Format                                |
+---------------------------------------------------------------------+
|                            ┌ REEL ┐        ⎧ NO REWIND ⎫            |
|   CLOSE     file-name-1    │      │  [WITH ⎨           ⎬  ]         |
|                            └ UNIT ┘        ⎩ LOCK      ⎭            |
|                                                                     |
|                            ┌ REEL ┐        ⎧ NO REWIND ⎫            |
|            [file-name-2    │      │  [WITH ⎨           ⎬ ] ] ...    |
|                            └ UNIT ┘        ⎩ LOCK      ⎭            |
+---------------------------------------------------------------------+
```
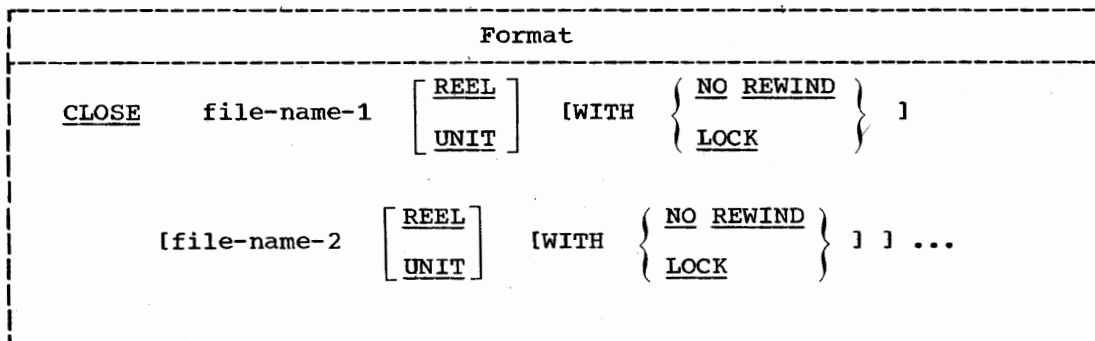

Each file-name is the name of a file upon which the CLOSE statement
is to operate; it must not be the name of a sort-file.   The file-name
must be defined by a file-description entry in the Data Division.

The REEL and WITH NO REWIND options are applicable only to tape
files.   The UNIT option is applicable only to mass storage files in
sequential access mode.

For purposes of showing the effect of various CLOSE options as
applied to various storage media, all input/output files are divided
into the following categories:

1.   Unit record volume. A file whose input or output medium is such
     that rewinding, units, and reels have no meaning.

2.   Sequential single-volume. A sequential file that is entirely
     contained on one volume. There may be more than one file on this
     volume.

3.   Sequential multivolume. A sequential file that may be contained on
     more than one volume.

4.   Random single-volume. A file in the random access mode that may be
     contained on a single mass storage volume.

5.   Random multivolume. A file in the random access mode that may be
     contained on more than one mass storage volume.

Sequential File Processing

The results of executing each CLOSE option for each type of file are summarized in Table 22. The definitions of the symbols in the illustration are given below. Where the definition of the symbol depends on whether the file is an input or output file, alternate definitions are given; otherwise, a definition applies to INPUT, OUTPUT, and I-O files.

A--Previous Volumes Unaffected

All volumes in the file prior to the current volume are processed according to standard volume switch procedures except those volumes controlled by a prior CLOSE REEL/UNIT statement.

B--No Rewind of Current Reel

The current volume is left in its current position.

C--Standard Close File

Files Opened as INPUT and I-O: If the file is positioned at its end and there is an ending label record, the standard ending label procedures and the user's ending label procedures (if specified by a USE sentence) are performed. System closing procedures are then performed.

If the file is positioned at its end and there is no ending label record, system closing procedures are performed.

If the file is positioned other than at its end, system closing procedures are performed but there is no ending label processing. An INPUT or an I-O file is considered to be at end-of-file if the imperative-statement in the AT END phrase has been executed and no CLOSE statement has been executed.

Files Opened as OUTPUT: If an ending label record has been described for the file, it is constructed and written on the output medium. System closing procedures are performed.

D--Standard Reel/Unit Lock

This feature has no meaning in this system and is treated as comments.

E--Standard File Lock

The compiler ensures that this file cannot be opened again during this execution of the object program.

CLOSE Statement

F--Standard Close Volume

Files Opened as INPUT and I-O: The following operations are performed:

1. A volume switch.

2. The standard beginning volume label procedure and the user's beginning volume label procedure (if specified by the USE sentence). The order of execution of these two procedures is specified by the USE sentence.

3. Makes the next data record on the new volume available to be read.

Files Opened as OUTPUT: The following operations are performed:

1. The standard ending volume label procedure and the user's ending volume label procedure (if specified by the USE statement). The order of execution of these two procedures is specified by the USE statement.

2. A volume switch.

3. The standard beginning volume label procedure and the user's beginning volume label procedure (if specified by the USE statement). The order of execution of these two procedures is specified by the USE statement.

G--Rewind

The current volume is positioned at its beginning.

X--Illegal

This is an illegal combination of a CLOSE option and a file type. The results at object time may be unpredictable.

Table 22.   Relationship of Types of Sequential Files and the Options of the CLOSE Statement

CLOSE Option \ FILE Type	Unit Record	Sequential Single-Volume	Sequential Multivolume
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, G
CLOSE REEL WITH NO REWIND	X	X	F, B
CLOSE UNIT	X	X	F
CLOSE UNIT WITH LOCK	X	X	F, D

General Considerations: A file is designated as optional by specifying the IGN parameter in the ASSGN control statement. If an optional file is not present, the standard end-of-file processing is not performed. For purposes of language consistency, the OPTIONAL phrase of the SELECT clause should be specified for this type of file.

If a CLOSE statement without the REEL or UNIT option has been executed for a file, the next input/output statement to be executed for that file must be an OPEN statement.

Random File Processing

The results of executing each CLOSE option for each type of file are summarized in Table 23. The definitions of the symbols in the figure are given below. Where the definition depends on whether the file is an INPUT or OUTPUT file, alternate definitions are given; otherwise, a definition applies to INPUT, OUTPUT, and I-O files.

H--Standard Close File

Files Opened as INPUT and I-O: If there is an ending label record, the ending label record is checked, and the conventional system closing procedures are performed. If there is no ending label record, the system closing procedures are performed. For I-O files, the label is updated and written.

Files Opened as OUTPUT: If an ending label record has been described for the file, it is constructed and written on the output medium. The system closing procedures are performed.

CLOSE Statement

J--<u>Standard File Lock</u>

The compiler ensures that this file cannot be opened again during this execution of the object program.

Table 23. Relationship of Types of Random Files and the Options of the CLOSE Statement.

CLOSE Option / FILE Type	Random Single-Volume	Random Multivolume
CLOSE	H	H
CLOSE WITH LOCK	H, J	H, J

Subprogram linkage statements are special statements that permit communication between object programs. These statements are CALL, ENTRY, GOBACK, and EXIT.

CALL Statement

The CALL statement permits communication between a COBOL object program and one or more COBOL subprograms or other language subprograms.

```
┌─────────────────────────────────────────────────────────────────────────┐
│                                 Format                                    │
├─────────────────────────────────────────────────────────────────────────┤
│                                                                           │
│    CALL literal [USING identifier-1 [identifier-2] ...]                    │
│                                                                           │
└─────────────────────────────────────────────────────────────────────────┘
```

Literal is a nonnumeric literal and is the name of the program that is being called, or the name of an entry point in the called program. The program in which the CALL statement appears is the calling program. Literal must conform to the rules for formation of a program-name. The first eight characters of literal are used to make the correspondence between the called and calling program.

When the called program is to be entered at the beginning of the Procedure Division, literal must specify the program-name in the PROGRAM-ID paragraph of the called program, and the called program must have a USING clause as part of its Procedure Division header if there is a USING clause in the CALL statement that invoked it.

When the called program is to be entered at entry points other than the beginning of the Procedure Division, these alternate entry points are identified by an ENTRY statement and a USING option corresponding to the USING option of the invoking CALL statement. In the case of a CALL statement with a corresponding ENTRY, literal must be a name other than the program-name but follows the same rules as those for the formation of a program-name.

The identifiers specified in the USING option of the CALL statement indicate those data items available to a calling program that may be referred to in the called program.

When the called subprogram is a COBOL program, each of the operands in the USING option of the calling program must be defined as a data item in the File Section, Working-Storage Section, or Linkage Section. If the called subprogram is written in a language other than COBOL, the operands of the USING option of the calling program may additionally be a file-name or a procedure-name. If the operand of the USING option is a file-name, the file with which the file-name is associated must be opened in the calling program.

Names in the two USING lists (that of the CALL in the main program and that of the Procedure Division header or the ENTRY in the subprogram) are paired in a one-for-one correspondence.

ENTRY Statement

There is no necessary relationship between the actual names used for such paired names, but the data descriptions must be equivalent. When a group data item is named in the USING list of a Procedure Division header or an ENTRY statement, names subordinate to it in the subprogram's Linkage Section may be employed in subsequent subprogram procedural statements.

When group items with level numbers other than 01 are specified, proper word-boundary alignment is required if subordinate items are described as COMPUTATIONAL, COMPUTATIONAL-1, or COMPUTATIONAL-2.

The USING option should be included in the CALL statement only if there is a USING option in the called entry point, which is either included in the Procedure Division header of the called program or included in an ENTRY statement in the called program. The number of operands in the USING option of the CALL statement should be the same as the number of operands in the USING option of the Procedure Division header, or ENTRY statement. If the number of operands in the USING option of the CALL statement is greater than the number in the USING option in the called program, only those specified in the USING option of the called program may be referred to by the called program.

The execution of a CALL statement causes control to pass to the called program. The first time a called program is entered, its state is that of a fresh copy of the program. Each subsequent time a called program is entered, the state is as it was upon the last exit from that program. Thus, the reinitialization of the following items is the responsibility of the programmer:

        GO TO statements which have been altered
        TALLY
        Data items
        ON statements
        PERFORM statements
        EXHIBIT CHANGED statements
        EXHIBIT CHANGED NAMED statements

    EXHIBIT CHANGED and EXHIBIT CHANGED NAMED operands will be compared against the value of the item at the time of its last execution, whether or not that execution was during another CALL to this program. If a branch is made out of the range of a PERFORM, after which an exit is made from the program, the range of that PERFORM is still in effect upon a subsequent entry.

Called programs may contain CALL statements. However, a called program must not contain a CALL statement that directly or indirectly calls the calling program.

A called program may not be segmented.

ENTRY Statement

The ENTRY statement establishes an entry point in a COBOL subprogram.

```
+----------------------------------------------------------------------+
|                               Format                                 |
+----------------------------------------------------------------------+
|                                                                      |
|    ENTRY literal [USING identifier-1 [identifier-2] ...]             |
|                                                                      |
+----------------------------------------------------------------------+
```
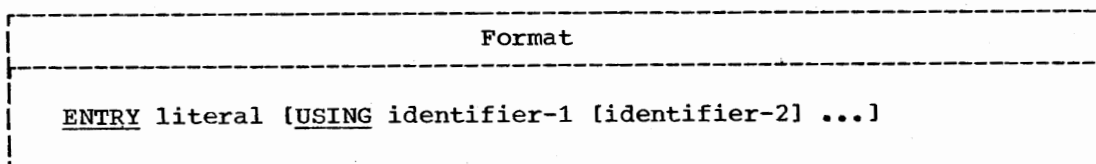
Control is transferred to the ENTRY point by a CALL statement in an invoking program.

Literal must not be the name of the called program, but is formed according to the same rules followed for program-names. Literal must not be the name of any other entry point or program-name in the run unit.

A called program, once invoked, is entered at that ENTRY statement whose operand, literal, is the same as the literal specified in the CALL statement that invoked it.

USING Option

The USING option makes data items defined in a calling program available to a called program. The number of operands in the USING option of a called program must be less than or equal to the number of operands in the corresponding CALL statement of the invoking program.

The USING option may be specified in the CALL statement, the ENTRY statement, or in the Procedure Division header.

```
r---------------------------------------------------------------------¬
|                     Format 1 (Within a Calling Program)             |
+---------------------------------------------------------------------+
|                                                                     |
|    CALL literal [USING identifier-1 [identifier-2]...]               |
|                                                                     |
L---------------------------------------------------------------------J
```

```
r---------------------------------------------------------------------¬
|                     Format 2 (Within a Called Program)              |
+---------------------------------------------------------------------+
|Option 1                                                             |
|                                                                     |
|    ENTRY literal [USING identifier-1 [identifier-2]...]              |
|                                                                     |
+---------------------------------------------------------------------+
|Option 2                                                             |
|                                                                     |
|    PROCEDURE DIVISION [USING identifier-1 [identifier-2]...].        |
|                                                                     |
L---------------------------------------------------------------------J
```

When the USING option is specified in the CALL statement, it must appear on either the Procedure Division header of the called program, or in an ENTRY statement in the called program.

The USING option may be present on the Procedure Division header or in an ENTRY statement, if the object program is to function under the control of a CALL statement, and the CALL statement contains a USING option.

When a called program has a USING option on its Procedure Division header and linkage was effected by a CALL statement where literal is the name of the called program, execution of the called program begins with the first instruction in the Procedure Division after the Declaratives Section.

USING Option

When linkage to a called program is effected by a CALL statement where _literal_ is the name of an entry point specified in the ENTRY statement of the called program, that execution of the called program begins with the first statement following the ENTRY statement.

Each of the operands in the USING option of the Procedure Division header or the ENTRY statement must have been defined as a data item in the Linkage Section of the program in which this header or ENTRY statement occurs, and must have a level number of 01 or 77. Since the compiler assumes that each level-01 item is aligned upon a double-word boundary, it is the programmer's responsibility to ensure proper alignment.

When the USING option is present, the object program operates as though each occurrence of _identifier-1_, _identifier-2_, etc., in the Procedure Division had been replaced by the corresponding identifier from the USING option in the CALL statement of the calling program. That is, corresponding identifiers refer to a single set of data which is available to the calling program. The correspondence is positional and not by name.

The following is an example of a calling program with the USING option:

```
    IDENTIFICATION DIVISION.
    PROGRAM-ID. CALLPROG.
        •
        •
        •
    DATA DIVISION.
        •
        •
        •
    WORKING-STORAGE SECTION.
    01  RECORD-1.
        03  SALARY      PICTURE S9(5)V99.
        03  RATE        PICTURE S9V99.
        03  HOURS       PICTURE S99V9.
        •
        •
        •
    PROCEDURE DIVISION.
        •
        •
        •
    CALL "SUBPROG" USING RECORD-1.
        •
        •
        •
    CALL "PAYMASTR" USING RECORD-1.
        •
        •
        •
```

The following is an example of a called subprogram associated with the preceding calling program:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SUBPROG.
    .
    .
    .
DATA DIVISION.
    .
    .
    .
LINKAGE SECTION.
01   PAYREC.
    02   PAY            PICTURE S9(5)V99.
    02   HOURLY-RATE    PICTURE S9V99.
    02   HOURS          PICTURE S99V9.
    .
    .
    .
PROCEDURE DIVISION USING PAYREC.
    .
    .
    .
    GOBACK.
    ENTRY "PAYMASTR" USING PAYREC.
    .
    .
    .
    GOBACK.
```

Processing begins in CALLPROG, which is the calling program. When the statement

```
    CALL "SUBPROG" USING RECORD-1.
```

is executed, control is transferred to the first statement of the Procedure Division in SUBPROG, which is the called program. In the calling program, the operand of the USING option is identified as RECORD-1.

When SUBPROG receives control, the values within RECORD-1 are made available to SUBPROG; in SUBPROG, however, they are referred to as PAYREC. Note that the PICTURE clauses for the subfields of PAYREC (described in the Linkage Section of SUBPROG) are the same as those for RECORD-1.

When processing within SUBPROG reaches the first GOBACK statement, control is returned to CALLPROG at the statement immediately following the original CALL statement. Processing then continues in CALLPROG until the statement

```
    CALL "PAYMASTR" USING RECORD-1.
```

is reached. Control is again transferred to SUBPROG, but this time processing begins at the statement following the ENTRY statement in SUBPROG. The values within RECORD-1 are again made available to SUBPROG through the matching USING operand PAYREC. When processing reaches the second GOBACK statement, control is returned to CALLPROG at the statement immediately following the second CALL statement.

USING Option

In any given execution of these two programs, if the values within RECORD-1 are changed between the time of the first CALL and the second, the values passed at the time of the second CALL statement will be the changed, not the original, values. If the programmer wishes to use the original values, then he must ensure that they have been saved.

Program Termination Considerations

There are three ways in COBOL source language to terminate a program. They are:

1. EXIT PROGRAM

2. GOBACK

3. STOP RUN

Table 24 shows the effect of each program termination statement based on whether it is issued within a main program or a subprogram.

A main program is the highest level COBOL program invoked in a step. A subprogram is a COBOL program that is invoked by another COBOL program. (Programs written in other languages that follow COBOL linkage conventions are considered COBOL programs in this sense.)

Table 24. Effect of Program Termination Statements Within Main Programs and Subprograms.

Termination Statement	Main Program	Subprogram
EXIT PROGRAM	Non-operational	Return to invoking program
STOP RUN	Return to system and cause end of job step (EOJ macro)	Return to system and cause end of job step (EOJ macro)
GOBACK	Abnormal termination of job	Return to invoking program

EXIT PROGRAM Statement

This form of the EXIT statement marks the logical end of a called program.

```
r--------------------------------------------------------------------------┐
|                              Format                                       |
|--------------------------------------------------------------------------|
|                                                                          |
|    paragraph-name.   EXIT PROGRAM.                                       |
|                                                                          |
L--------------------------------------------------------------------------┘
```

The EXIT statement must be preceded by a paragraph-name and be the only statement in the paragraph.

If control reaches an EXIT PROGRAM statement while operating under the control of a CALL statement, control returns to the point in the calling program immediately following the CALL statement.

If control reaches an EXIT PROGRAM statement and no CALL statement is active, control passes through the exit point to the first sentence of the next paragraph.

GOBACK Statement

The GOBACK statement marks the logical end of a called program.

```
r--------------------------------------------------------------------------┐
|                              Format                                       |
|--------------------------------------------------------------------------|
|                                                                          |
|    GOBACK.                                                               |
|                                                                          |
L--------------------------------------------------------------------------┘
```

A GOBACK statement must appear as the only statement or as the last of a series of imperative-statements in a sentence.

If control reaches a GOBACK statement while operating under the control of a CALL statement, control returns to the point in the calling program immediately following the CALL statement.

If control reaches a GOBACK statement and no CALL statement is active, there will be an abnormal termination of the job.

STOP RUN Statement

For a discussion of the STOP statement with the RUN option, see "Procedure Branching Statements."

COMPILER-DIRECTING STATEMENTS

Compiler directing statements are special statements that provide instructions for the COBOL compiler. The compiler directing statements are COPY, ENTER, and NOTE.

COPY Statement

Prewritten source program entries can be included in a COBOL program at compile time. Thus, an installation can utilize standard file descriptions, record descriptions, or procedures without having to repeat programming them. These entries and procedures are contained in user-created libraries. They are included in a source program by means of a COPY statement (see "Source Program Library Facility").

ENTER Statement

The ENTER statement serves only as documentation and is intended to provide a means of allowing the use of more than one source language in the same source program. This compiler allows no other source language in the program.

```
┌──────────────────────────────────────────────────────────────────┐
│                              Format                                │
├──────────────────────────────────────────────────────────────────┤
│                                                                    │
│    ENTER language-name [routine-name].                             │
│                                                                    │
└──────────────────────────────────────────────────────────────────┘
```

The ENTER statement is accepted as comments.

NOTE Statement

The NOTE statement allows the programmer to write commentary which will be produced on the source listing, but not compiled.

```
┌──────────────────────────────────────────────────────────────────┐
│                              Format                                │
├──────────────────────────────────────────────────────────────────┤
│                                                                    │
│    NOTE character string                                           │
│                                                                    │
└──────────────────────────────────────────────────────────────────┘
```

Any combination of the characters from the EBCDIC set may be included in the character string.

If a NOTE sentence is the first sentence of a paragraph, the entire paragraph is considered to be part of the <u>character string</u>. Proper format rules for paragraph structure must be observed.

If a NOTE sentence appears as other than the first sentence of a paragraph, the commentary ends with the first instance of a period followed by a space.

Explanatory comments may be inserted on any line within a source program by placing an asterisk in column 7 of the line. Any combination of the characters from the EBCDIC set may be included in Area A and Area B of that line. The asterisk and the characters will be produced on the listing, but serve no other purpose.
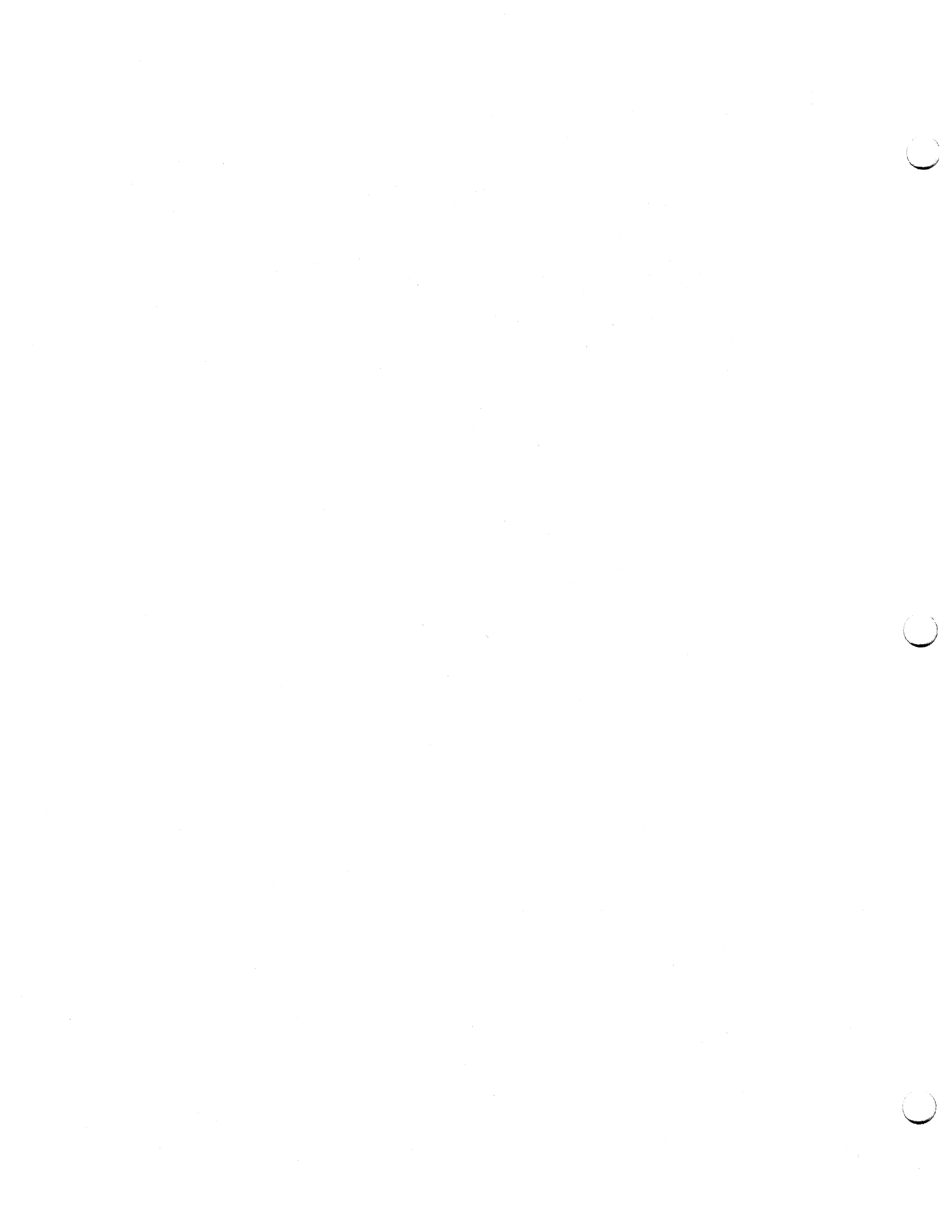
- SORT FEATURE

- REPORT WRITER FEATURE

- TABLE HANDLING FEATURE

- SEGMENTATION FEATURE

- SOURCE PROGRAM LIBRARY FACILITY

- DEBUGGING LANGUAGE

- FORMAT CONTROL OF THE SOURCE PROGRAM LISTING

- STERLING CURRENCY FEATURE
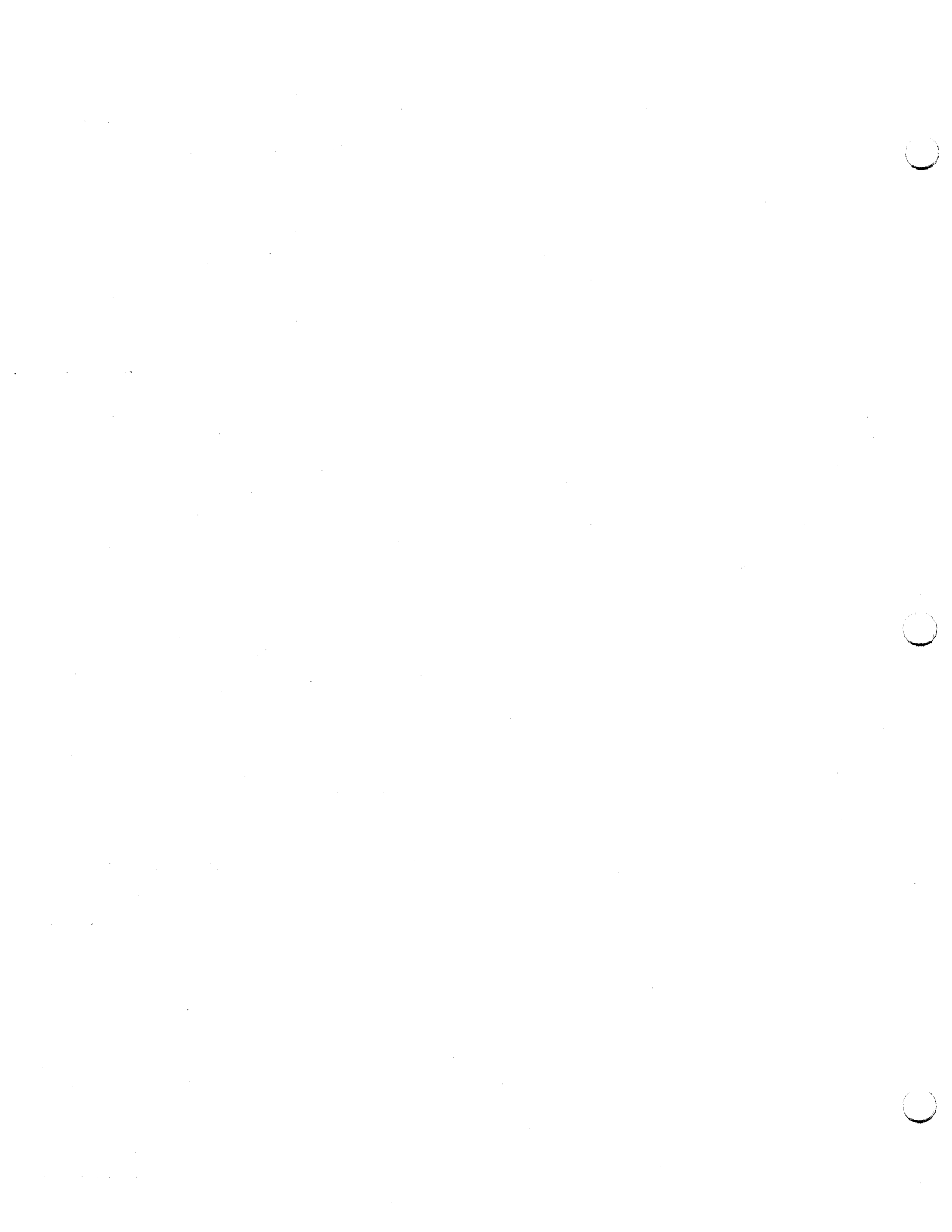
The COBOL programmer can gain convenient access to the sorting capability of the system sort/merge program by including a SORT statement and other elements of the Sort Feature in his source program. The Sort Feature provides the capability for sorting files and including procedures for special handling of these files both before and after they have been sorted. Within the limits of object-time storage, a source program may have any number of SORT statements, and each SORT statement may have its own special procedures.

The basic elements of the Sort Feature are the SORT statement in the Procedure Division and the Sort-File-Description (SD) entry, with its associated record description entries, in the Data Division. A sorting operation is based on sort-keys named in the SORT statement. A sort-key specifies the field within a record on which the file is sorted. Sort-keys are defined in the record description associated with the SD entry. The records of a file may be sorted in ascending or descending order or in a mixture of the two; that is, the sort-keys may be specified as ascending or descending, independent of one another, and the sequence of the sorted records will conform to the mixture specified. Additional information on the Sort Feature can be found in the publication IBM System/360 Disk Operating System: American National Standard COBOL Programmer's Guide, Form GC28-6398.

Program Product Information -- Version 3

Information on the Program Product Version 3 Sort Feature can be found in the publication IBM System/360 Disk Operating System: Full American National Standard COBOL Compiler and Library Version 3 Programmer's Guide, Order No. SC28-6441.

Note: Language considerations for an ASCII-collated sort are given in Appendix E.

ELEMENTS OF THE SORT FEATURE

To use the Sort Feature, the COBOL programmer must provide additional information in the Environment, Data, and Procedure Divisions of the source program.

The SORT statement in the Procedure Division is the primary element of a source program that performs one or more sorting operations. The term "sorting operation" means not only the manipulation by the Sort Program of sort-work-files on the basis of the sort-keys designated by the COBOL programmer, but also includes the method of making records available to, and retrieving records from, these sort-work-files. A sort-work-file is the collection of records that is involved in the sorting operation as it exists on an intermediate device(s). Records are made available either by the USING or INPUT PROCEDURE options of the SORT statement. Sorted records are retrieved either by the GIVING or OUTPUT PROCEDURE options of the SORT statement.

In the Environment Division, the programmer must write SELECT sentences for all files used as input and output to the Sort Program and

for the sort-file. If checkpoint records are to be taken during the sorting operation, he must also include a RERUN statement.

In the Data Division, the programmer must include File Description entries (FD) for all files that are used to provide input to or output from the sort program and for the sort file. He must also write a Sort-File-Description (SD) entry and its associated record description entries to describe the records that are to be sorted, including their sort-key fields.

In the Procedure Division SORT statement, the programmer specifies the sort-file to be sorted, and the sort-key names. He must also specify whether the sort is to be in ascending or descending sequence by key, and whether records are to have special processing. If there is to be such processing, he also includes in the Procedure Division the program sections that perform the processing. Special SORT registers, if used, are referenced in the Procedure Division.

ENVIRONMENT DIVISION CONSIDERATIONS FOR SORT

There are certain statements the programmer must include in the Environment Division to use the Sort Feature. Detailed descriptions of these statements follow.

INPUT-OUTPUT SECTION

The Input-Output Section is composed of two parts: the File-Control Paragraph and the I-O-Control Paragraph.

File-Control Paragraph

The File-Control Paragraph is specified once in a COBOL program. Within this paragraph, all files referred to in the source program must be named in a SELECT clause.

Files used within input and output procedures and files named in the USING and GIVING options of the SORT statement are named in the SELECT clause as described in "Environment Division." The file named in the GIVING option of the SORT statement can alternately be described in the following format.

```
+----------------------------------------------------------------------+
|                              Format                                  |
+----------------------------------------------------------------------+
|                                                                      |
|   SELECT file-name                                                   |
|                                                                      |
|        ASSIGN TO [integer-1] system-name-1 [system-name-2] ...       |
|                                          ( REEL )                    |
|        OR system-name-3 [FOR MULTIPLE    {      }  ]                 |
|                                          ( UNIT )                    |
|                     ( integer-2 )              ( AREA  )             |
|        [RESERVE     {          }  ALTERNATE    {       }  ].         |
|                     ( NO        )              ( AREAS )             |
|                                                                      |
+----------------------------------------------------------------------+
```

The OR option is neither required nor used by this compiler, and is treated as comments.
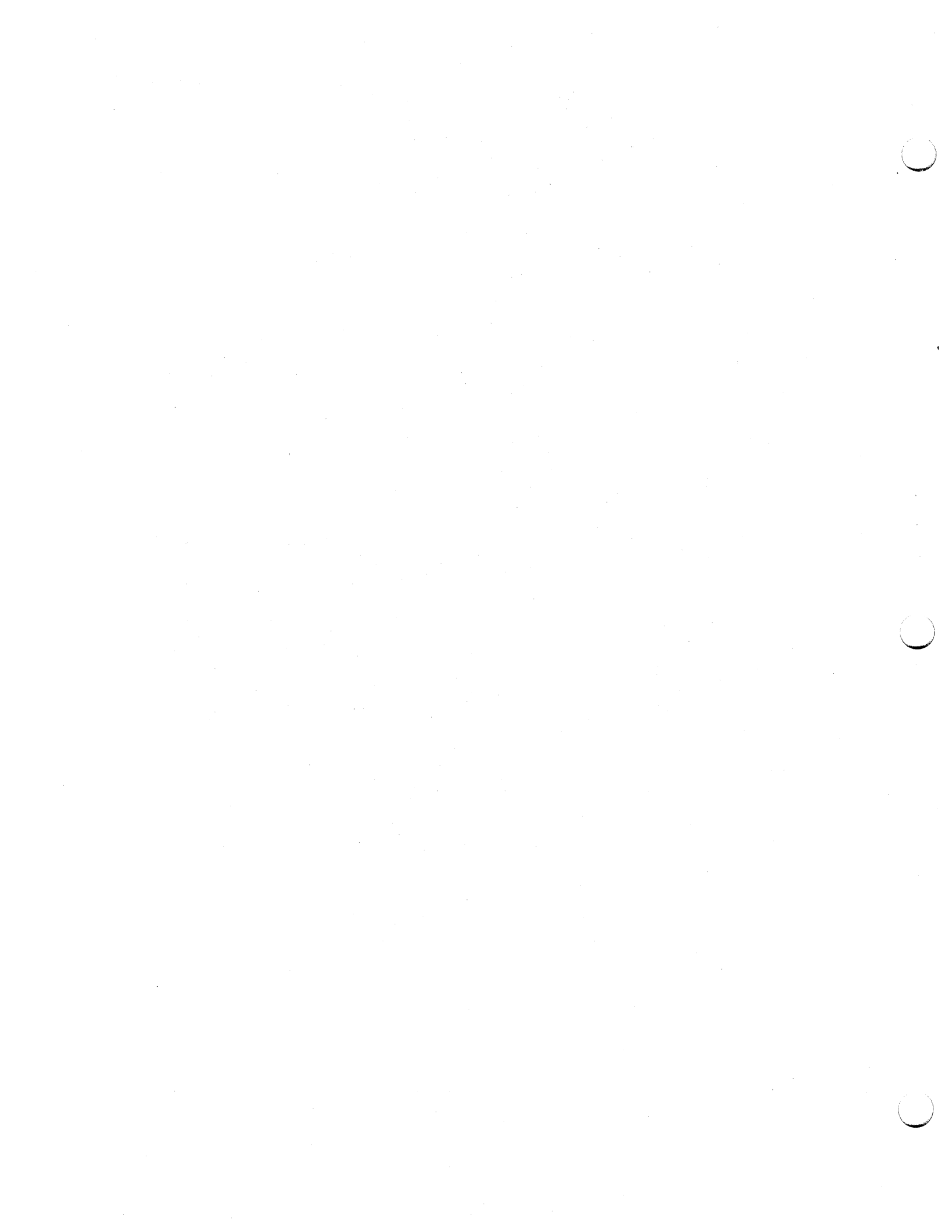
The MULTIPLE clause function is specified by object time control cards; hence, the MULTIPLE clause is neither required nor used by this compiler. The RESERVE clause is applicable as described in the "Environment Division" chapter.

Assignment of Sort Work Units

The File Control paragraph must be specified for the sort file and is used to assign work units for the sorting operation.

```
r-------------------------------------------------------------------------------
|                                     Format                                    |
|-------------------------------------------------------------------------------|
|                                                                               |
|   SELECT sort-file-name                                                       |
|                                                                               |
|       ASSIGN TO [integer] system-name-1 [system-name-2] ...                   |
|                                                                               |
L-------------------------------------------------------------------------------
```

sort-file-name: the name used as the first operand of the SORT statement (also the name associated with the SD entry for the sorting operation).

integer: specifies the number of work units available to the Sort Program. From one through eight units may be assigned for a disk sort. From three through nine units may be assigned for a tape sort. If integer is not specified, the compiler assumes one unit for a disk sort and three units for a tape sort.

system-name-1: created in the same format as the system-name in other ASSIGN clauses (see "Environment Division"). However, the names by which the work files are known to the Sort Program are fixed. The first work unit for every sort-file in the program must be assigned to SYS001, the second to SYS002, etc. If the files have standard labels, the name field must be specified, SORTWK1 for SYS001, SORTWK2 for SYS002, etc.

Integer work units beginning with the first are reserved for the sorting operation. The user may, if he wishes, specify these additional work units in multiple system-names. However, the compiler treats these as comments.

For example, the SELECT sentence for a sort-file with standard labels, which has five work units (tape) available, would be:

    SELECT SORTFILE
        ASSIGN TO 5 SYS001-UT-2400-S-SORTWK1.

SYS001 through SYS005 are assigned by the compiler to the work units.

I-O-CONTROL Paragraph

The I-O-Control paragraph specifies when checkpoints are to be taken, as well as what core storage area is to be shared by different files. The I-O-Control paragraph is coded once in the source program. The

checkpoint interval associated with the standard RERUN format (specified in the "Environment Division") is determined by the number of records processed for the given file. However, the format has meaning only when a file is not being used in a sorting operation. Obtaining checkpoint records within the operation of a SORT statement is specified by a special format of the RERUN statement, as described below.

RERUN Clause

   The format of the RERUN clause used in conjunction with the sorting operation is:

```
-----------------------------------------------------------------------
|                              Format                                  |
|---------------------------------------------------------------------|
|                                                                     |
|    RERUN ON system-name                                             |
|                                                                     |
-----------------------------------------------------------------------
```

   The presence of this format of the RERUN clause indicates that checkpoint records are to be written, at logical intervals determined by the sort program, during the execution of all SORT statements that appear in a source program. Its absence indicates that, within the execution of any SORT statement, checkpoint records are not to be taken.

   System-name must not be the same as any system-name used in a FILE-CONTROL ASSIGN clause, but must follow the same rules of formation. This system-name may be the same as the system name specified in a standard RERUN clause.

   At the time the checkpoint procedure of the SORT statement takes effect, the status of all open files, whether involved in the sorting operation or not, is recorded.

SAME RECORD/SORT AREA Clause

   The SAME RECORD/SORT AREA clause specifies that two or more files are to use the same storage area during processing.

```
-----------------------------------------------------------------------
|                              Format                                  |
|---------------------------------------------------------------------|
|                                                                     |
|           ( RECORD )                                                |
|    SAME   {        }   AREA     FOR file-name-1 {file-name-2} ...    |
|           ( SORT   )                                                |
|                                                                     |
-----------------------------------------------------------------------
```

   When the RECORD option is used, the named files, including any sort-files, share only the area in which the current logical record is processed. Several of the files may be open at the same time, but the logical record of only one of these files can exist in the record area at one time.

The function of the SORT option is to optimize the assignment of storage areas to a given SORT statement. The system handles storage assignment automatically; hence, the SORT option, if given, is treated as comments.

DATA DIVISION CONSIDERATIONS FOR SORT

In the Data Division the programmer must include File Description entries for files that are input to or output from the Sort, sort-file-description entries which describe the records as they appear on the sort work files, and record description entries for each.

FILE SECTION

The File Section of a program which contains a sorting operation must furnish information concerning the physical structure, identification, and record names of the sort work file. This is provided in the sort-file-description entry.

Sort-File Description

A sort-file-description entry must appear in the File Section for every file named as the first operand of a SORT statement.


```
+------------------------------------------------------------------------+
|                                Format                                  |
+------------------------------------------------------------------------+
|                                                                        |
|   SD  sort-file-name                                                    |
|                                                                        |
|          [RECORDING MODE IS mode]                                      |
|                                                                        |
|                        (  RECORD IS   )                                |
|              [DATA     {              }  data-name-1 [data-name-2] ...] |
|                        ( RECORDS ARE  )                                |
|                                                                        |
|          [RECORD CONTAINS [integer-1 TO] integer-2 CHARACTERS]         |
|                        ( RECORD IS   )  ( STANDARD )                   |
|              [LABEL    {              }  {          }  ].              |
|                        ( RECORDS ARE  )  ( OMITTED  )                   |
|                                                                        |
+------------------------------------------------------------------------+
```


There must be a SELECT sentence for sort-file-name that describes the system-name for the sort work file. Sort-file-name is also the name specified in the SORT statement.

The RECORDING MODE clause is discussed in "Data Division." The recording mode must be F or V.

The DATA RECORDS clause specifies the names of the records in the file to be sorted. Data-name-1, data-name-2, ... of the DATA RECORDS clause refer to the records described in the record descriptions associated with this SD.

The RECORD CONTAINS clause specifies the size of data records in the file to be sorted. This clause is optional. The actual size and mode (fixed or variable) of the records to be sorted are determined from the level-01 descriptions associated with a given SD entry. When the USING and GIVING options of the SORT statement are used, the record length associated with the SD must be the same length as the record associated with the FD's for the USING and GIVING files. If any of the SD data record descriptions contains an OCCURS clause with the DEPENDING ON option, variable-length records are assumed. Refer to "Data Division" for the format assumptions that are made by the compiler when the RECORDING MODE clause is not specified.

Both the DATA RECORDS and the RECORD CONTAINS clauses are described in "Data Division."

The LABEL RECORDS clause specifies whether the sort-work-files have standard labels or no labels. If this clause is not specified, OMITTED is assumed.

PROCEDURE DIVISION CONSIDERATIONS FOR SORT

The Procedure Division must contain a SORT statement to describe the sorting operation and, optionally, input and output procedures. The procedure-names constituting the input and output procedures are specified within the SORT statement.

The Procedure Division may contain more than one SORT statement appearing anywhere except in the declaratives portion or in the input and output procedures associated with a SORT statement.

SORT Statement

The SORT statement provides information that controls the sorting operation. This information directs the sorting operation to obtain records to be sorted either from an input procedure or the USING file to sort the records on a set of specified sort keys, and in the final phase of the sorting operation to make each record available in sorted order, either to an output procedure or to the GIVING file.

```
+-----------------------------------------------------------------------+
|                               Format                                  |
+-----------------------------------------------------------------------+
|                                                                       |
|                            ( DESCENDING )                             |
|   SORT file-name-1 ON      {           }  KEY {data-name-1} ...        |
|                            ( ASCENDING  )                             |
|                                                                       |
|              ( DESCENDING )                                           |
|        [ON   {           }   KEY {data-name-2} ...] ...               |
|              ( ASCENDING  )                                           |
|                                                                       |
|        ( INPUT PROCEDURE IS section-name-1 [THRU section-name-2] )    |
|        { USING file-name-2                                       }    |
|                                                                       |
|        ( OUTPUT PROCEDURE IS section-name-3 [THRU section-name-4] )   |
|        { GIVING file-name-3                                       }   |
|                                                                       |
+-----------------------------------------------------------------------+
```

File-name-1 is the name given in the sort-file-description entry that describes the records to be sorted.

ASCENDING and DESCENDING: The ASCENDING and DESCENDING options specify whether records are to be sorted into an ascending or descending sequence, respectively, based on one or more sort keys.

Each data-name represents a "key" data item and must be described in the records associated with the sort-file-name.

At least one ASCENDING or DESCENDING clause must be specified. Both options may be specified in the same SORT statement, in which case, records are sorted on data-name-1, in ascending or descending order, and then within data-name-1, they are sorted on the KEY data item represented by data-name-2, in ascending or descending order, etc.

Keys are always listed from left to right in order of decreasing significance, regardless of whether they are ascending or descending.

The direction of the sort depends on the use of the ASCENDING or DESCENDING clauses as follows:

1. When an ASCENDING clause is used, the sorted sequence is from the lowest value of the key to the highest value, according to the collating sequence for the COBOL character set.

2. When a DESCENDING clause is used, the sorted sequence is from the highest value of the key to the lowest value, according to the collating sequence of the COBOL character set.

Sort keys must be one of the types of data item listed in Figure 13. Corresponding to each type of data item is a collating sequence that is used with it for sorting.

A character in the EBCDIC collating sequence (used with alphabetic, alphanumeric, etc., data items) is interpreted as not being signed. For fixed-point and internal floating-point numeric data items, characters are collated algebraically (that is, as being signed).

Type of Data Item Used for Sort Key	Collating Sequence
Alphabetic	EBCDIC
Alphanumeric	EBCDIC
Numeric Edited	EBCDIC
Group	EBCDIC
External Decimal	Zoned Decimal
Binary	Fixed Point
Internal Decimal	Fixed point
Internal Floating point	Floating Point
External Floating point	EBCDIC

Figure 13. SORT Collating Sequences Used for Sort Keys

The EBCDIC collating sequence for COBOL characters in ascending order is:

1.		(space)
2.	.	(period or decimal point)
3.	<	(less than)
4.	(	(left parenthesis)
5.	+	(plus symbol)
6.	$	(currency symbol)
7.	*	(asterisk)
8.	)	(right parenthesis)
9.	;	(semicolon)
10.	-	(hyphen or minus symbol)
11.	/	(stroke, virgule, slash)
12.	,	(comma)
13.	>	(greater than)
14.	'	(apostrophe or single quotation mark)
15.	=	(equal sign)
16.	"	(quotation mark)
17-42.	A through Z	
43-52.	0 through 9	

(The complete EBCDIC collating sequence is given in IBM System/360 Reference Data, Form X20-1703.)

The record description for every record that is listed in the DATA RECORDS clause of an SD description must contain the "key" items data-name-1, data-name-2, etc. These "key" items are subject to the following rules:

1. Keys must be physically located in the same position and have the same data format in every logical record of the sort-file. If there are multiple record descriptions in an SD, it is sufficient to describe a key in only one of the record descriptions.

2. Key items must not contain an OCCURS clause nor be subordinate to entries that contain an OCCURS clause.

3. A maximum of 12 keys may be specified. The total length of all the keys must not exceed 256 bytes.

4. All keys must be at a fixed displacement from the beginning of a record; that is, they cannot be located following a variable table in a record.

5. All key fields must be located within the first 4092 bytes of a logical record.

6. The data-names describing the keys may be qualified.

SECTION-NAME-1 AND SECTION-NAME-2: Section-name-1 is the name of an input procedure. Section-name-2 is the name of the last section that contains the input procedure in the COBOL main program. Section-name-2 is required if the procedure terminates in a section other than that in which it was started.

INPUT PROCEDURE: The presence of the INPUT PROCEDURE option indicates that the programmer has written an input procedure to process records before they are sorted and has included the procedure in the Procedure Division as one or more distinct sections.

The input procedure must consist of one or more sections that are written consecutively and do not form a part of any output procedure. The input procedure must include at least one RELEASE statement in order to transfer records to the sort-file.

Control must not be passed to the input procedure unless a related SORT statement is being executed, because the RELEASE statement in the input procedure has no meaning unless it is controlled by a SORT statement. The input procedure can include any procedures needed to select, create, or modify records. There are three restrictions on the procedural statements within an input procedure:

1. The input procedure must not contain any SORT statements.

2. The input procedure must not contain any transfers of control to points outside the input procedure. The execution of USE declaratives for label handling and error processing are not considered transfers of control outside of an input procedure. Hence, they are allowed to be activated within these procedures.

> However, this compiler permits the ALTER, GO TO, and PERFORM statements in the input procedure to refer to procedure-names outside the input procedure. It is the user's responsibility to insure a return to the input procedure after exiting through a GO TO or PERFORM statement.

3. The remainder of the Procedure Division must not contain any transfers of control to points inside the input procedure (with the exception of the return of control from a declarative section).

> However, this compiler allows ALTER, GO TO, and PERFORM statements in the remainder of the Procedure Division to refer to precedure-names within the Input Procedure. If a SORT statement is active when the transfer of control is made, then all such transfers are valid. If a SORT statement is not active, however, then the user must ensure that such a transfer of control does not cause:
>
> a. a RELEASE statement to be executed
>
> b. control to reach the end of the Input Procedure

If an input procedure is specified, control is passed to the input procedure when the SORT program input phase is ready to receive the first record. The compiler inserts a return mechanism at the end of the last section of the input procedure and, when control passes the last statement in the input procedure, the records that have been released to file-name-1 are sorted.

The RELEASE statement transfers records from the Input Procedure to the input phase of the sort operation (see "RELEASE Statement").

USING: If the USING option is specified, all the records in file-name-2 are transferred automatically to file-name-1. At the time of execution of the SORT statement, file-name-2 must not be open. File-name-2 must be a standard sequential file.

For the USING option, the compiler will open, read, release, and close file-name-2 without the programmer specifying these functions. If the user specifies error handling and/or label processing declaratives for file-name-2, the compiler will make the necessary linkage to the appropriate declarative section.

SECTION-NAME-3 AND SECTION-NAME-4: Section-name-3 represents the name of an output procedure. Section-name-4 is the name of the last section that contains the output procedure in the COBOL main program. Section-name-4 is required if the procedure terminates in a section other than that in which it is started.

OUTPUT PROCEDURE: The output procedure must consist of one or more
sections that are written consecutively and do not form a part of any
input procedure. The output procedure must include at least one RETURN
statement in order to make sorted records available for processing.

Control must not be passed to the output procedure unless a related
SORT statement is being executed, because RETURN statements in the
output procedure have no meaning unless they are controlled by a SORT
statement. The output procedure may consist of any procedures needed to
select, modify, or copy the records that are being returned one at a
time, in sorted order, from the sort-file. There are three restrictions
on the procedural statements within the output procedure.

1. The output procedure must not contain any SORT statements.

2. The output procedure must not contain any transfers of control to
    points outside the output procedure. The execution of USE
    declaratives for label handling and error processing are not
    considered transfers of control outside of an output procedure.
    Hence, they are allowed to be activated within these procedures.

    However, this compiler permits the ALTER, GO TO, and PERFORM
    statements in the output procedure to refer to procedure-names
    outside the output procedure. It is the user's responsibility to
    insure a return to the output procedure after exiting through a GO
    TO or PERFORM statement.

3. The remainder of the Procedure Division must not contain any
    transfers of control to points inside the output procedure (with
    the exception of the return of control from a declarative section).

    However, this compiler allows ALTER, GO TO, and PERFORM statements
    in the remainder of the Procedure Division to refer to
    procedure-names within the Output Procedure. If a SORT statement
    is active when the transfer of control is made, then all such
    transfers are valid. If a SORT statement is not active, however,
    then the user must ensure that such a transfer of control does not
    cause:

    a. a RETURN statement to be executed

    b. control to reach the end of the Output Procedure

If an output procedure is specified, control passes to it after
file-name-1 has been placed in sequence by the SORT statement. The
compiler inserts a return mechanism at the end of the last section in
the output procedure. When control passes the last statement in the
output procedure, the return mechanism provides for termination of the
SORT and then passes control to the next statement after the SORT
statement.

When all the records are sorted, control is passed to the output
procedure. The RETURN statement in the output procedure is a request
for the next record (see "RETURN Statement").

GIVING: If the GIVING option is used, all sorted records in file-name-1
are automatically transferred to file-name-3. At the time of execution
of the SORT statement, file-name-3 must not be open. File-name-3 must
name a standard sequential file.

For the GIVING option, the compiler will open, return, write, and
close file-name-3 without the programmer specifying these functions. If
the user specifies error handling and/or label processing declaratives

for file-name-3, the compiler will make the necessary linkage to the appropriate declarative section.

CONTROL OF INPUT OR OUTPUT PROCEDURES: The INPUT or OUTPUT PROCEDURE options function in a manner similar to the PERFORM statement (Option 1); for example, naming a section in an INPUT PROCEDURE clause causes execution of that section during the sorting operation to proceed as though that section had been the subject of a PERFORM statement. As in the execution of a PERFORM statement, the execution of the section is terminated after execution of its last statement. The procedure may be terminated by an EXIT statement (see "EXIT Statement").

RELEASE Statement

The RELEASE statement transfers records from the Input Procedure to the input phase of the Sort operation.

```
┌─────────────────────────────────────────────────────────────────────┐
│                             Format                                    │
├─────────────────────────────────────────────────────────────────────┤
│                                                                       │
│    RELEASE sort-record-name [FROM identifier]                         │
│                                                                       │
└─────────────────────────────────────────────────────────────────────┘
```

A RELEASE statement may be used only within the range of an input procedure associated with a SORT statement.

If the INPUT PROCEDURE option is specified, the RELEASE statement must be included within the given set of procedures.

Sort-record-name must be the name of a logical record in the associated sort-file description.

When the FROM identifier option is used, it makes the RELEASE statement equivalent to the statement MOVE identifier TO sort-record-name, followed by the statement RELEASE.

Sort-record-name and identifier must not refer to the same storage area. A move with the rules for group items is effected from identifier, using the length of the record-name associated with the SD entry.

RETURN/EXIT Statements

The RETURN statement obtains individual records in sorted order from the final phase of the sort program.

```
,---------------------------------------------------------------------------,
|                                Format                                     |
|---------------------------------------------------------------------------|
|                                                                           |
|    RETURN sort-file-name RECORD [INTO identifier]                         |
|                                                                           |
|        AT END imperative-statement                                        |
|                                                                           |
'---------------------------------------------------------------------------'
```

Sort-file-name is the name given in the sort-file-description entry that describes the records to be sorted.

All references to records retrieved by a RETURN statement must be in terms of the record description(s) associated with the SD entry, unless the INTO option is specified. The retrieved record may, optionally, be moved to the user's own area and be referenced as appropriate.

A RETURN statement may only be used within the range of an output procedure associated with a SORT statement for file-name-1.

The identifier must be the name of a working-storage area or an output record area. Use of the INTO option has the same effect as the MOVE statement for alphanumeric items.

The imperative-statement in the AT END phrase specifies the action to be taken when all the sorted records have been obtained from the sorting operation.

EXIT Statement

The EXIT statement may be used as a common end point for input or output procedures as with procedures executed through a PERFORM statement.

```
,---------------------------------------------------------------------------,
|                                Format                                     |
|---------------------------------------------------------------------------|
|                                                                           |
|    paragraph-name.  EXIT.                                                 |
|                                                                           |
'---------------------------------------------------------------------------'
```

When used in this manner, the EXIT statement must appear as the only statement in the last paragraph of the input or output procedure.

SPECIAL REGISTERS FOR SORT

Four special registers are available to users of the Sort Feature. These registers provide a means of object time communication between the user and the Sort Feature, and they aid in the optimization of sort performance.

The first three registers may have control information transferred to them at object time if the user specifies them as the receiving fields of statements such as MOVE. However, none of the Sort registers can be used as operands in ACCEPT, DISPLAY, or EXHIBIT statements. The information must be passed before the SORT statement is executed. The registers are initialized to zero by the compiler, but are not reset after a sort procedure is executed. Thus, if a program has multiple SORT statements, any values in the registers at the completion of one sorting operation will be in the registers at the beginning of another, unless modified.

1. SORT-FILE-SIZE is the name of a binary data item whose PICTURE is S9(8). It is used for the estimated number of records in the file to be sorted. If SORT-FILE-SIZE is omitted, the Sort Feature assumes that the file contains the maximum number of records that can be processed with the available core size and number of work units. If the estimate exceeds the maximum, the estimate will be ignored.

2. SORT-CORE-SIZE is the name of a binary data item whose PICTURE is S9(8). It is used to specify the number of bytes of storage available to the sorting operation if it is different from the core size that the Sort Feature would normally use.

3. SORT-MODE-SIZE is the name of a binary data item whose PICTURE is S9(5). It is used for variable-length records. If the length of most records in the file is significantly different from the average record length, performance is improved by specifying the most frequently occurring record length. If SORT-MODE-SIZE is omitted, the average length is assumed. For example, if records vary in length from 20 to 100 bytes, but most records are 30 bytes long, the value 30 should be moved to SORT-MODE-SIZE. The maximum record length handled by the Sort Feature is 32,000 bytes.

4. SORT-RETURN is the name of a binary data item whose PICTURE is S9(4). It contains a return code of 0 or 16 at the completion of a sorting operation to signify the success or failure, respectively, of the sort operation.

An example of how variable information can be passed to the Sort Feature by use of a register is:

```
ACCEPT FILE-SIZE FROM CONSOLE.
MOVE FILE-SIZE TO SORT-FILE-SIZE.
```

SAMPLE PROGRAM USING THE SORT FEATURE

This example illustrates a sort based on a sales contest. The records to be sorted contain data on salesmen: name and address, employee number, department number, and pre-calculated net sales for the contest period.

The salesman with the highest net sales in each department wins a prize, and smaller prizes are awarded for second highest sales, third highest, etc. The order of the SORT is (1) by department, the lowest numbered first (ASCENDING KEY DEPT); and (2) by net sales within each department, the highest net sales first (DESCENDING KEY NET-SALES).

The records for the employees of departments 7 and 9 are eliminated in an input procedure (SCREEN-DEPT) before sorting begins. The remaining records are then sorted, and the output is placed on another file for use in a later job step.

```
000005 IDENTIFICATION DIVISION.
000010 PROGRAM-ID. CONTEST.
000015 ENVIRONMENT DIVISION.
000016 CONFIGURATION SECTION.
000017 SOURCE-COMPUTER.  IBM-360-F50.
000018 OBJECT-COMPUTER.  IBM-360-F50.
000019 SPECIAL-NAMES. SYSLST IS PRINTER.
000020 INPUT-OUTPUT SECTION.
000025 FILE-CONTROL.
000030     SELECT NET-FILE-IN ASSIGN TO SYS008-UT-2400-S.
000035     SELECT NET-FILE-OUT ASSIGN TO SYS007-UT-2400-S-SORTOUT.
000040     SELECT NET-FILE ASSIGN TO 3 SYS001-UT-2400-S.
000050 DATA DIVISION.
000055 FILE SECTION.
000060 SD   NET-FILE
000065      DATA RECORD IS SALES-RECORD.
000070 01   SALES-RECORD.
000075      02   EMPL-NO        PICTURE 9(6).
000080      02   DEPT           PICTURE 9(2).
000085      02   NET-SALES      PICTURE 9(7)V99.
000090      02   NAME-ADDR      PICTURE X(55).
000095 FD   NET-FILE-IN
000096      LABEL RECORDS ARE OMITTED
000100      DATA RECORD IS NET-CARD-IN.
000105 01   NET-CARD-IN.
000110      02   EMPL-NO-IN     PICTURE 9(6).
000115      02   DEPT-IN        PICTURE 9(2).
000120      02   NET-SALES-IN   PICTURE 9(7)V99.
000125      02   NAME-ADDR-IN   PICTURE X(55).
000130 FD   NET-FILE-OUT
000131      LABEL RECORDS ARE OMITTED
000135      DATA RECORD IS NET-CARD-OUT.
000140 01   NET-CARD-OUT.
000145      02   EMPL-NO-OUT    PICTURE 9(6).
000150      02   DEPT-OUT       PICTURE 9(2).
000155      02   NET-SALES-OUT  PICTURE 9(7)V99.
000160      02   NAME-ADDR-OUT  PICTURE X(55).
```

Figure 14.   Sample Program Using the SORT Feature (Part 1 of 2)

```
|000165 PROCEDURE DIVISION.
|000170 ELIM-DEPT-7-9-NO-PRINTOUT.
|000175     SORT NET-FILE
|000180         ASCENDING KEY DEPT,
|000185         DESCENDING KEY NET-SALES
|000190         INPUT PROCEDURE SCREEN-DEPT
|000195         GIVING NET-FILE-OUT.
|000200 CHECK-RESULTS SECTION.
|000205 C-R-1.
|000210     OPEN INPUT NET-FILE-OUT.
|000215 C-R-2.
|000220     READ NET-FILE-OUT AT END GO TO C-R-FINAL.
|000225     DISPLAY EMPL-NO-OUT DEPT-OUT NET-SALES-OUT
|000230         NAME-ADDR-OUT UPON PRINTER.
|000235 C-R-3.
|000240     GO TO C-R-2.
|000245 C-R-FINAL.
|000250     CLOSE NET-FILE-OUT.
|000255     STOP RUN.
|000260 SCREEN-DEPT SECTION.
|000265 S-D-1.
|000270     OPEN INPUT NET-FILE-IN.
|000275 S-D-2.
|000280     READ NET-FILE-IN AT END GO TO S-D-FINAL.
|000285     DISPLAY EMPL-NO-IN DEPT-IN NET-SALES-IN
|000290         NAME-ADDR-IN UPON PRINTER.
|000295 S-D-3.
|000300     IF DEPT-IN = 7 OR 9 GO TO S-D-2
|000305         ELSE
|000310             MOVE NET-CARD-IN TO SALES-RECORD,
|000315             RELEASE SALES-RECORD,
|000320             GO TO S-D-2.
|000325 S-D-FINAL.
|000330     CLOSE NET-FILE-IN.
|000335 S-D-END.
|000340     EXIT.
```

Figure 14. Sample Program Using the SORT Feature (Part 2 of 2)

REPORT WRITER FEATURE

The Report Writer Feature permits the programmer to specify the format of a printed report in the Data Division, thereby minimizing the amount of Procedure Division coding he would have to write to create the report.

A printed report consists of the information reported and the format in which it is printed. Several reports can be produced by one program.

In the Data Division, the programmer gives the name(s) and describes the format(s) of the report(s) he wishes produced. In the Procedure Division, he writes the statements that produce the report(s).

At program execution time, the report in the format defined is produced -- data to be accumulated is summed, totals are produced, counters are stepped and reset, and each line and each page is printed. Thus, the programmer need not concern himself with the details of these operations.

DATA DIVISION -- OVERALL DESCRIPTION

In the Data Division, the programmer must write an FD entry that names the output file upon which the report is to be written, and must also name the report itself. A report may be written on two files at the same time.

At the end of the Data Division, he must add a Report Section to define the format of each report named. In the Report Section, there are two types of entries:

1. The Report Description Entry (RD) which describes the physical aspects of the report format.

2. The report group description entries which describe the data items within the report and their relation to the report format.

In the report description entry, the programmer specifies the maximum number of lines per page, where report groups are to appear on the page, and which data items are to be considered as controls.

Controls govern the basic format of the report. When a control changes value -- that is, when a control break occurs -- special actions will be taken before the next line of the report is printed. Controls are listed in a hierarchy, proceeding from the most inclusive down to the least inclusive. Thus, by specifying HEADING and FOOTING controls, the programmer is able to instruct the Report Writer to produce the report in whatever format he desires.

For example, in the program at the end of this chapter, the hierarchy of controls proceeds from the highest (FINAL) to an intermediate control (MONTH) to the minor control (DAY). DAY is the minor control since, if MONTH changes, DAY also must change. Whenever any control changes, special actions are performed by the Report Writer -- sum information is totaled, counters are reset, special information is printed, and so forth -- before the next detail line is printed.

The report group description entries describe the characteristics of all data items contained within the report group: the format of each data item present, its placement in relation to the other data items within the report group, and any control factors associated with the

group. Information to be presented within a report group can be
described in three ways:

* as <u>SOURCE information</u>, which is information from outside the report

* as <u>SUM information</u>, which is the result of addition operations upon
  any data present, whether SOURCE information or other SUM
  information

* as <u>VALUE information</u>, which is constant information

   Through the RD and the report group description entries, the
programmer has thus defined completely the content, the format, and the
summing operations necessary to produce the desired report.

PROCEDURE DIVISION -- OVERALL DESCRIPTION

   In the Procedure Division, the programmer instructs the Report Writer
to produce the report through the use of three Report Writer statements:
INITIATE, GENERATE, and TERMINATE.

   The INITIATE statement performs functions in the Report Writer
analogous to the OPEN statement for individual files.

   The GENERATE statement automatically produces the body of the report.
Necessary headings and footings are printed, counters are incremented
and reset as desired, source information is obtained, and sum
information is produced, data is moved to the data item(s) in the report
group description entry, controls are tested, and when a control break
occurs, the additional lines requested are printed, as well as the
detail line that caused the control break. All of this is done
automatically, thus relieving the programmer of the responsibility for
writing detailed tests and looping procedures that would otherwise be
necessary.

   The TERMINATE statement completes the processing of a report. It is
analogous to the CLOSE statement for individual files.

   In the Declaratives portion of the Procedure Division, the programmer
may also specify a USE BEFORE REPORTING procedure for report group. In
this procedure, he is able to specify any additional processing he
wishes done before a specific report group is printed.

   Two special registers are used by the Report Writer feature:

LINE-COUNTER -- which is a numeric counter used by the Report Writer to
        determine when a PAGE HEADING and/or a PAGE FOOTING report group is
        to be presented. The maximum value of LINE-COUNTER is based on the
        number of lines per page as specified in the PAGE LIMIT(S) clause.
        LINE-COUNTER may be referred to in any Procedure Division
        statement.

PAGE-COUNTER -- which is a numeric counter that may be used as a SOURCE
        data item in order to present the page number on a report line.
        The maximum size of PAGE-COUNTER is based on the size specified in
        the PICTURE clause associated with an elementary item whose SOURCE
        IS PAGE-COUNTER. This counter may be referred to by any Procedure
        Division statement.

   Figure 16, at the end of this chapter, gives an example of a Report
Writer program for a manufacturer's quarterly report.

FD Entry/REPORT Clause

Figure 17, which follows the program, shows the report that would be produced.

DATA DIVISION CONSIDERATIONS FOR REPORT WRITER

The names of all the reports to be produced must be named in the File Section of the Data Division. An entry is required in the FD entry to list the names of the reports to be produced on that file. A Report Section must be added at the end of the Data Division to define the format of each report.

FILE DESCRIPTION

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

```
+-----------------------------------------------------------------+
|                         General Format                          |
+-----------------------------------------------------------------+
|                                                                 |
|   FD file-name                                                  |
|                                                                 |
|        [BLOCK CONTAINS Clause]                                  |
|        [RECORD CONTAINS Clause]                                 |
|        [RECORDING MODE Clause]                                  |
|        LABEL RECORDS Clause                                     |
|        [VALUE OF Clause]                                        |
|        [DATA RECORDS Clause]                                    |
|        REPORT Clause.                                           |
|                                                                 |
+-----------------------------------------------------------------+
```

A discussion of all the above-mentioned clauses appears in "Data Division." A description of the REPORT clause, the RECORDING MODE clause, the DATA RECORDS clause, and the RECORD CONTAINS clause for a file on which a report is produced follows.

REPORT Clause

Each unique report-name must appear in the REPORT clause of the FD entry (or entries) for the file(s) on which the report(s) is to be produced. The REPORT clause cross references the description of Report Description entries with their associated File Description entry.

```
+-----------------------------------------------------------------+
|                            Format                               |
+-----------------------------------------------------------------+
|                                                                 |
|    { REPORT IS   }                                              |
|    {             }   report-name-1 [report-name-2]...           |
|    { REPORTS ARE }                                              |
|                                                                 |
+-----------------------------------------------------------------+
```

RECORD CONTAINS Clause

Each File Description entry for standard sequential OUTPUT files within the File Section may include a REPORT clause containing the names of one or more reports. These reports may be of different sizes, formats, etc., and the order in which their names appear in the clause is not significant.

Each unique report-name listed in an FD entry must be the subject of an RD entry in the Report Section. A given report-name may appear in a maximum of two REPORT clauses.

RECORDING MODE Clause

The RECORDING MODE clause is used to specify the format of the logical records within the file. If this clause is omitted, the recording mode is determined as described in "Data Division."

DATA RECORDS Clause

If the DATA RECORDS clause is specified, and the file is used for output, the AFTER ADVANCING option must be used when writing records named in this clause.

RECORD CONTAINS Clause

The RECORD CONTAINS clause enables the user to specify the maximum size of his report record.

```
┌─────────────────────────────────────────────────────────────────────────┐
│                                 Format                                    │
├─────────────────────────────────────────────────────────────────────────┤
│                                                                           │
│    RECORD CONTAINS [integer-1 TO] integer-2 CHARACTERS                     │
│                                                                           │
└─────────────────────────────────────────────────────────────────────────┘
```

The specified size of each report record must include the carriage control/line spacing character, and the CODE character, if the CODE option is used. If the RECORD CONTAINS clause is omitted, the compiler assumes a default size of 133 characters.

For variable-length records, the size of each print line will be integer-2 characters, and the size of each blank line required for spacing will be 17 characters. For fixed-length records, the size of each print line and each blank line required for spacing will be integer-2 characters.

For further information on the RECORD CONTAINS clause, see "Data Division."

RD (report description) Entry

REPORT SECTION

The Report Section consists of two types of entries for each report; one describes the physical aspects of the report format, the other type describes conceptual characteristics of the items that make up the report and their relationship to the report format. These are:

1.   Report Description entry (RD)

2.   Report group description entries

The Report Section must begin with the header REPORT SECTION.

Report Description Entry

The Report Description entry contains information pertaining to the overall format of a report named in the File Section and is uniquely identified by the level indicator RD. The clauses that follow the name of the report are optional, and their order of appearance is not significant.

The entries in this section stipulate:

1.   The maximum number of lines that can appear on a page.

2.   Where report groups are to appear on a page.

3.   Data items that act as control factors during presentation of the report.

```
r------------------------------------------------------------------1
|                          General Format                          |
|------------------------------------------------------------------|
|                                                                  |
|    REPORT SECTION.                                               |
|                                                                  |
|    RD report-name                                                |
|        [CODE Clause]                                             |
|        [CONTROL Clause]                                          |
|        [PAGE LIMIT Clause].                                      |
|                                                                  |
L------------------------------------------------------------------J
```

RD is the level indicator.

Report-name is the name of the report and must be unique. The report-name must be specified in a REPORT clause in the File Description entry for the file on which the report is to be written.

CODE Clause

The CODE clause is used to specify an identifying character added at the beginning of each line produced. The identification is meaningful when more than one report is written on a file.

```
r--------------------------------------------------------------------¬
¦                              Format                                ¦
¦-------------------------------------------------------------------¦
¦                                                                    ¦
¦    WITH CODE mnemonic-name                                         ¦
¦                                                                    ¦
L--------------------------------------------------------------------
```

Mnemonic-name must be associated with a single character literal used
as function-name-1 in the SPECIAL-NAMES paragraph in the Environment
Division. The identifying character is appended to the beginning of the
line, preceding the carriage control/line spacing character. This
clause should not be specified if the report is to be printed on-line.

CONTROL Clause

     The CONTROL clause indicates the identifiers that specify the control
hierarchy for this report, that is, the control breaks.


```
r--------------------------------------------------------------------¬
¦                              Format                                ¦
¦-------------------------------------------------------------------¦
¦                                                                    ¦
¦ ( CONTROL IS    )  ( FINAL                                     )   ¦
¦ {               }  {  identifier-1 [identifier-2]...           }   ¦
¦ ( CONTROLS ARE  )  ( FINAL   identifier-1 [identifier-2]...    )   ¦
¦                                                                    ¦
L--------------------------------------------------------------------
```

     A control is a data item that is tested each time a detail report
group is generated. If the test indicates that the value of the data
item (i.e., CONTROL) has changed, a control break is said to occur, and
special action (described below) is taken before printing the detail
line.

     FINAL is the highest level control. (It is the one exception to the
statement that controls are data items.) The identifiers specify the
control hierarchy of the other controls. Identifier-1 is the major
control, identifier-2 is the intermediate control, etc. The last
identifier specified is the minor control. The levels of the controls
are indicated by the order in which they are written. FINAL need not be
specified in the CONTROL clause, even if a CONTROL HEADING FINAL or
CONTROL FOOTING FINAL is specified.

     When controls are tested, the highest level control specified is
tested first, then the second highest level, etc. When a control break
is found for a particular level, a control break is implied for each
lower level as well. A control break for FINAL occurs only at the
beginning and ending of a report (i.e., before the first detail line is
printed and after the last detail is printed).

     The action to be taken as a result of a control break depends on what
the programmer defines. He may define a CONTROL HEADING report group
and/or a CONTROL FOOTING group or neither for each control.

     The control footings and headings that are defined are printed prior
to printing the originally referenced detail. They are printed in the
following order: lowest level control footing, next higher level
control footing, etc., up to and including the control footing for the

level at which the control break occurred; then the control heading for
that level, then the next lower level control heading, etc., down to and
including the minor control heading; then the detail is printed. If, in
the course of printing control headings and footings, a page condition
is detected, the current page is ejected and a new page begun. If the
associated report groups are given, a page footing and/or a page heading
are also printed.

The CONTROL clause is required when CONTROL HEADING or CONTROL
FOOTING report groups other than FINAL are specified.

The identifiers specified in the CONTROL clause are the only
identifiers referred to by the RESET and TYPE clauses in a report group
description entry for this report. The identifiers must be defined in
the File or Working-Storage Section of the Data Division.

PAGE LIMIT Clause

The PAGE LIMIT clause indicates the specific line control to be
maintained within the logical presentation of a page, i.e., it describes
the physical format of a page of the report.

```
┌──────────────────────────────────────────────────────────────────────┐
│                              Format                                    │
├──────────────────────────────────────────────────────────────────────┤
│                                                                        │
│         ┌  LIMIT IS   ┐                 ⎧ LINE  ⎫                       │
│  PAGE   │             │   integer-1     ⎨       ⎬                       │
│         └  LIMITS ARE ┘                 ⎩ LINES ⎭                       │
│                                                                        │
│      [HEADING integer-2]                                               │
│      [FIRST DETAIL integer-3]                                          │
│      [LAST DETAIL integer-4]                                           │
│      [FOOTING integer-5]                                               │
│                                                                        │
└──────────────────────────────────────────────────────────────────────┘
```

If this clause is not specified, PAGE-COUNTER and LINE-COUNTER
special registers are not generated.

The PAGE LIMIT clause is required when page format must be controlled
by the Report Writer.

integer-1:   The PAGE LIMIT integer-1 LINES clause is required to
                specify the depth of the report page; the depth of the
                report page may or may not be equal to the physical
                perforated continuous form often associated in a report
                with the page length. The size of the fixed data-name,
                LINE-COUNTER, is the maximum numeric size based on
                integer-1 lines required for the counter to prevent
                overflow.

integer-2:   The first line number of the first heading print group is
                specified by integer-2. No print group will start
                preceding integer-2, i.e., integer-2 is the first line on
                which anything may be printed.

integer-3:   The first line number of the first normal print group
                (body group) is specified by integer-3. No DETAIL,
                CONTROL HEADING, or CONTROL FOOTING print group will start
                before integer-3.

<u>integer-4</u>: The last line number of the last nonfooting body group is specified by integer-4. No DETAIL or CONTROL HEADING print group will extend beyond integer-4.

<u>integer-5</u>: The last line number of the last CONTROL FOOTING print group is specified by integer-5. No CONTROL FOOTING print group will extend beyond integer-5. PAGE FOOTING print groups will follow integer-5.

Using the parameters of the PAGE LIMIT clause, the Report Writer establishes the areas of the page where each type of report group is allowed to be printed. The following are the page areas for each type of report group:

1. A REPORT HEADING report group can extend from line integer-2 to line integer-1, inclusive. If the REPORT HEADING report group is not on a page by itself, the FIRST DETAIL integer-3 clause must be present in the PAGE LIMIT clause of the report.

2. A PAGE HEADING report group may extend from line integer-2 to line integer-3 minus 1, inclusive. If a PAGE HEADING report group is specified in the report description, the FIRST DETAIL integer-3 clause must be present in the PAGE LIMIT clause of the report. A PAGE HEADING report group that follows a REPORT HEADING report group on the same page must be able to be printed in the area of the page defined in this rule.

3. CONTROL HEADING report groups and DETAIL report groups must be printed in the area of the page that extends from line integer-3 to line integer-4, inclusive.

4. CONTROL FOOTING report groups must be printed in the area of the page extending from line integer-3 to line integer-5, inclusive.

5. A PAGE FOOTING report group may extend from line integer-5 plus 1 to line integer-1, inclusive. If PAGE FOOTING is specified in the report description, either the FOOTING integer-5 or LAST DETAIL integer-4 clause must be present in the PAGE LIMIT clause of the report.

6. A REPORT FOOTING report group can extend from line integer-2 to line integer-1, inclusive. If the REPORT FOOTING report group is not on a page by itself, either the FOOTING integer-5 or LAST DETAIL integer-4 clause must be present in the PAGE LIMIT clause of the report.

Figure 15 pictorially represents page format report group control when the PAGE LIMIT clause is specified.
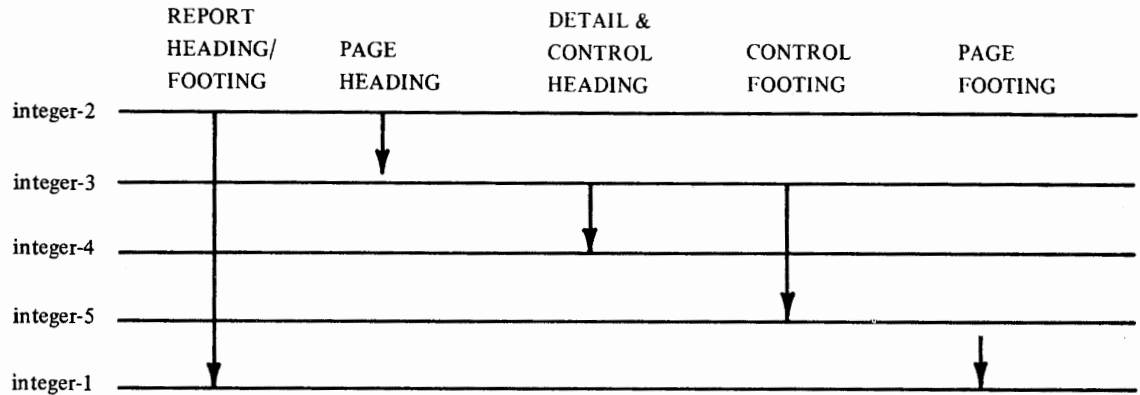
Figure 15.   Page Format when the PAGE LIMIT Clause Is Specified

The PAGE LIMIT clause may be omitted when no association is desired between report groups and the physical format of an output page. In this case, relative line spacing must be indicated for all report groups of the report.

If absolute line spacing is indicated for all the report groups, none of the integer-2 through integer-5 controls need be specified. If any of these limits are specified for a report that has only absolute line spacing, the limits are ignored.

If relative line spacing is indicated for any report group, all LINE NUMBER and NEXT GROUP spacing must be consistent with the controls specified or implied in the PAGE LIMIT clause.

If PAGE LIMITS integer-1 is specified and some or all of the HEADING integer-2, FIRST DETAIL integer-3, LAST DETAIL integer-4, FOOTING integer-5 clauses are omitted, the following implicit control is assumed for all omitted specifications:

1.   If HEADING integer-2 is omitted, integer-2 is considered to be equivalent to the value 1, that is, LINE NUMBER one.

2.   If FIRST DETAIL integer-3 is omitted, integer-3 is considered to be equivalent to the value of integer-2.

3.   If LAST DETAIL integer-4 is omitted, integer-4 is considered to be equivalent to the value of integer-5.

4.   If FOOTING integer-5 is omitted, integer-5 is considered to be equivalent to the value of integer-4. If both LAST DETAIL integer-4 and FOOTING integer-5 are omitted, integer-4 and integer-5 are both considered to be equivalent to the value of integer-1.

Only one PAGE-LIMIT clause may be specified for a Report Group Description entry.

- Integer-1 through integer-5 must be positive integers.

- Integer-2 through integer-5 must be in ascending order. Integer-5 must not exceed integer-1.

Report Group Description Entry

A report comprises one or more report groups. Each report group is described by a hierarchy of entries similar to the description of a data record. There are three categories of report groups: heading groups, detail groups, and footing groups. A CONTROL HEADING, DETAIL, or CONTROL FOOTING report group may also be referred to as a body group.

The report group description entry defines the format and characteristics for a report group. The relative placement of a particular report group within the hierarchy of report groups, the format of all items, and any control factors associated with the group are defined in this entry.

Schematically, a report group is a line, a series of lines, or a null (i.e., nonprintable) group. A report group is considered to be one unit of the report. Therefore, the lines of a report group are printed as a unit. A null group is a report group for which no LINE clauses are specified.

The report group description entry defines the format and characteristics applicable to the type of report group.

1. For all report groups that are not null, the description entry indicates where and when the report group is to be presented.

2. For all report groups, the description entry indicates when the nonprinting functions of the report group, such as summation, are to be performed.

3. For all report groups except DETAIL, the description entry allows for the execution of a user-specified procedure prior to printing a report group. If a report group is null, the execution of the user procedure occurs in the same manner as though the report group were printed.

4. For CONTROL FOOTING report groups, the description entry indicates the user's summation algorithm.

Report group names are required when reference is made in the Procedure Division:

- to a DETAIL report group by a GENERATE statement.

- to a HEADING or FOOTING report group by a USE sentence.

Report group names are required when reference is made in the Report Section to a DETAIL report group by a SUM UPON clause.

Except for the data-name clause which, when present, must immediately follow the level number, the clauses may be written in any order.

Report Group Description Entry--Formats

```
r------------------------------------------------------------------------¬
|                          General Format 1                              |
+------------------------------------------------------------------------+
|                                                                        |
|   01    [data-name-1]                                                  |
|         [LINE Clause]                                                   |
|         [NEXT GROUP Clause]                                            |
|         TYPE Clause                                                     |
|         [USAGE Clause].                                                 |
|                                                                        |
L------------------------------------------------------------------------
```

```
r------------------------------------------------------------------------¬
|                          General Format 2                              |
+------------------------------------------------------------------------+
|                                                                        |
|   level number [data-name-1]                                           |
|         [LINE clause]                                                   |
|         [USAGE clause].                                                 |
|                                                                        |
L------------------------------------------------------------------------
```

```
r------------------------------------------------------------------------¬
|                          General Format 3                              |
+------------------------------------------------------------------------+
|                                                                        |
|   level number [data-name-1]                                           |
|         [BLANK WHEN ZERO Clause]                                       |
|         [COLUMN Clause]                                                 |
|         [GROUP Clause]                                                  |
|         [JUSTIFIED Clause]                                             |
|         [LINE Clause]                                                   |
|         [PICTURE Clause]                                                |
|         [RESET Clause]                                                  |
|              ( SOURCE )                                                 |
|         [ {  SUM    }  Clause    ]                                      |
|              ( VALUE  )                                                 |
|         [USAGE Clause].                                                 |
|                                                                        |
L------------------------------------------------------------------------
```

```
r------------------------------------------------------------------------¬
|                          General Format 4                              |
+------------------------------------------------------------------------+
|                                                                        |
|   01    [data-name-1]                                                  |
|         [BLANK WHEN ZERO Clause]                                       |
|         [COLUMN Clause]                                                 |
|         [GROUP Clause]                                                  |
|         [JUSTIFIED Clause]                                             |
|         [LINE Clause]                                                   |
|         [NEXT GROUP Clause]                                            |
|         PICTURE Clause                                                  |
|         [RESET Clause]                                                  |
|           ( SOURCE )                                                    |
|           {  SUM    }  Clause                                           |
|           ( VALUE  )                                                    |
|         TYPE Clause                                                     |
|         [USAGE Clause].                                                 |
|                                                                        |
L------------------------------------------------------------------------
```

Format 1 is used to indicate a report group. A report group description must contain a report group entry (level-01) and it must be the first entry. A report group extends from this entry either to the next report group level-01 entry or to the end of the next report description. A null report group may contain only a Format 1 report group entry.

Format 2 is used to indicate a group item. A group item entry may contain a level number from 02 through 49; this entry has the following functions:

- If a report group has more than one line and one of the lines contains more than one elementary item, a group item entry may be used to indicate the LINE number of the subordinate elementary items.

- If a group item entry contains no LINE clause and there are no SUM counters subordinate to it, its only function is documentation.

Format 3 is used to indicate an elementary item. An elementary item entry may contain a level number from 02 through 49; this entry has the following functions:

- An elementary item entry may be used to describe an item that is to be presented on a printed line. In this case, a COLUMN clause, a PICTURE clause, and either a SOURCE, SUM, or VALUE clause must be present.

- An elementary item entry in a DETAIL report group may be used to indicate to the Report Writer what operands are to be summed upon presentation of the DETAIL report group.

- An elementary item entry in a CONTROL FOOTING report group may be used to define a SUM counter. (See SUM Clause.)

Format 4 is used to indicate a report group that consists of only one elementary item. If Format 4 is used to define the report group instead of Format 1, it must be the only entry in the group.

LINE Clause

The LINE clause indicates the absolute or relative line number of this entry in reference to the page or previous entry.

```
+------------------------------------------------------------------+
|                             Format                               |
+------------------------------------------------------------------+
|                        ( integer-1        )                      |
|    LINE NUMBER IS      { PLUS integer-2    }                      |
|                        ( NEXT PAGE        )                       |
|                                                                  |
+------------------------------------------------------------------+
```

Each line of a report must have a LINE clause associated with it. For the first line of a report group, the LINE clause must be given either at the report group level or prior to or for the first elementary item in the line. For report lines other than the first in a report group, the LINE clause must be given prior to or for the first

elementary item in the line. When a LINE clause is encountered,
subsequent entries following the entry with the LINE clause are
implicitly presented on the same line until either another LINE clause
or the end of the report group is encountered.

Integer-1 and integer-2 must be positive integers.

LINE NUMBER IS integer-1 is an absolute LINE clause. It indicates
the fixed line of the page on which this line is to be printed.
LINE-COUNTER is set to the value of integer-1 and is used for printing
the items in this and the following entries within the report group
until a different value for the LINE-COUNTER is specified.

LINE NUMBER IS PLUS integer-2 is a relative LINE clause. The line is
printed relative to the previous line either printed or skipped.
LINE-COUNTER is incremented by the value of integer-2 and is used for
printing the items in this and the following entries within the report
group until a different value for the LINE-COUNTER is specified.
Exceptions to this rule are discussed later.

LINE NUMBER IS NEXT PAGE indicates that this report group is to be
printed on the next page, not on the current page. This LINE clause may
appear only in a report group entry or may be the LINE clause of the
first line of the report group.

Within any report group, absolute LINE NUMBER entries must be
indicated in ascending order, and an absolute LINE NUMBER cannot be
preceded by a relative LINE NUMBER. If the first line of the first body
group that is to be printed on a page contains either a relative LINE
clause or a LINE NUMBER IS NEXT PAGE clause, the line is printed on line
FIRST DETAIL integer-3. However, if the LINE-COUNTER contains a value
that is greater than or equal to FIRST DETAIL integer-3, the line is
printed on line LINE-COUNTER plus 1. This value of LINE-COUNTER was set
by an absolute NEXT GROUP clause in the previously printed body group
(see rules for NEXT GROUP).

If the report group entry of a body group contains a LINE NUMBER IS
NEXT PAGE clause and the first line contains a relative LINE clause, the
first line is printed relative to either FIRST DETAIL integer-3 or
LINE-COUNTER, whichever is greater. This value of LINE-COUNTER was set
by an absolute NEXT GROUP clause in the previously printed body group.

The following are the rules for the LINE clause by report group type:

1. REPORT HEADING

    • LINE NUMBER IS NEXT PAGE cannot be specified in the report group.

    • The first line of the report group may contain an absolute or
      relative LINE clause.

    • If the first line contains a relative line clause, it is relative
      to HEADING integer-2.

2. PAGE HEADING

    • LINE NUMBER IS NEXT PAGE cannot be specified in the report group.

    • The first line may contain either an absolute or relative LINE
      clause.

    • If the first line contains a relative LINE clause, it is relative
      to either HEADING integer-2 or the value of LINE-COUNTER,
      whichever is greater. The value in LINE-COUNTER that is greater
      than HEADING integer-2 can only result from a REPORT HEADING

report group being printed on the same page as the PAGE HEADING
report group.

3. CONTROL HEADING, DETAIL, and CONTROL FOOTING

    • LINE NUMBER IS NEXT PAGE may be specified in the report group.

    • The first line of the report group may contain either an absolute
      or relative LINE clause.

4. PAGE FOOTING

    • LINE NUMBER IS NEXT PAGE cannot be specified in the report group.

    • The first line of the report group may contain an absolute or
      relative LINE clause.

    • If the first line contains a relative LINE clause, it is relative
      to FOOTING integer-5.

5. REPORT FOOTING

    • If the report group is to be printed on a page by itself, LINE
      NUMBER IS NEXT PAGE must be specified.

    • If LINE NUMBER IS NEXT PAGE is the only LINE clause in the report
      group description, the line will be printed on line HEADING
      integer-2.

    • If the report group description does not contain a LINE NUMBER IS
      NEXT PAGE clause, the first line must contain an absolute or
      relative LINE clause. If it contains a relative LINE clause, the
      line is relative to either FOOTING integer-5 or the value of
      LINE-COUNTER, whichever is greater. The value in LINE-COUNTER
      that is greater than FOOTING integer-5 can only result from the
      printing of the PAGE FOOTING report group.

NEXT GROUP Clause

The NEXT GROUP clause indicates the spacing condition following the
last line of the report group.

```
+---------------------------------------------------------------------+
|                              Format                                  |
+---------------------------------------------------------------------+
|                                                                     |
|                           (  integer-1       )                      |
|        NEXT GROUP IS      {  PLUS integer-2   }                      |
|                           (  NEXT PAGE        )                      |
|                                                                     |
+---------------------------------------------------------------------+
```

The NEXT GROUP clause can appear only in a report group entry.
Integer-1 and integer-2 must be positive integers.

NEXT GROUP Clause

The following are the rules for the NEXT GROUP clause by report group type:

1. REPORT HEADING

    •   If the report group is to be printed on a page by itself, NEXT GROUP IS NEXT PAGE must be specified in the report group description.

    •   Integer-1 indicates an absolute line number which sets the LINE-COUNTER to this value after printing the last line of the report group.

    •   Integer-2 indicates a relative line number which increments the LINE-COUNTER by the integer-2 value after printing the last line of the report group.

    •   An absolute or relative NEXT GROUP clause must not cause the LINE-COUNTER to be set to a value greater than FIRST DETAIL integer-3 minus 1.

2. PAGE HEADING, PAGE FOOTING, and REPORT FOOTING

    •   A NEXT GROUP clause cannot be specified in the report group.

3. CONTROL HEADING, DETAIL, and CONTROL FOOTING

    •   If a NEXT GROUP clause implies a page change, the change occurs only when the next body group is to be printed.

    •   The NEXT GROUP IS NEXT PAGE clause indicates that no more body groups are to be printed on this page.

    •   An absolute or relative NEXT GROUP clause may cause the LINE-COUNTER to be set to a value greater than or equal to FIRST DETAIL integer-3 and less than or equal to FOOTING integer-5. This is an exception to the rule which defines the page area of CONTROL HEADING and DETAIL report groups.

    •   If a NEXT GROUP IS integer-1 clause causes a page change, the value of LINE-COUNTER is set to the value of integer-1 before the formatting of the first line of the next body group to be printed. This implies that if the first line of the next body group to be printed contains a relative LINE NUMBER clause, the line will be printed on line LINE-COUNTER plus 1; if the first line contains an absolute LINE NUMBER clause that is less than or equal to integer-1, a page will be printed which contains only PAGE HEADING and FOOTING report groups, and the following page will contain the body group.

    •   When the NEXT GROUP clause is specified for a CONTROL FOOTING report group, the NEXT GROUP clause functions are performed only when a control break occurs for the control that is associated with this report group.

If the USE procedure for a report group contains a MOVE 1 PRINT-SWITCH statement, the NEXT GROUP clause functions are not performed for this report group.

The TYPE clause specifies the particular type of report group that is described by this entry and indicates the time at which the report group is to be generated.

```
.-----------------------------------------------------------------------.
|                                Format                                  |
|-----------------------------------------------------------------------|
|            / { REPORT HEADING }                        \               |
|            | { RH            }                         |               |
|            | { PAGE HEADING  }                         |               |
|            | { PH            }                         |               |
|            | { CONTROL HEADING}      { identifier-n}   |               |
|            | { CH            }       { FINAL       }    |               |
| TYPE IS    < { DETAIL        }                         >               |
|            | { DE            }                         |               |
|            | { CONTROL FOOTING}      { identifier-n}   |               |
|            | { CF            }       { FINAL       }    |               |
|            | { PAGE FOOTING  }                         |               |
|            | { PF            }                         |               |
|            | { REPORT FOOTING}                         |               |
|            \ { RF            }                         /               |
'-----------------------------------------------------------------------'
```
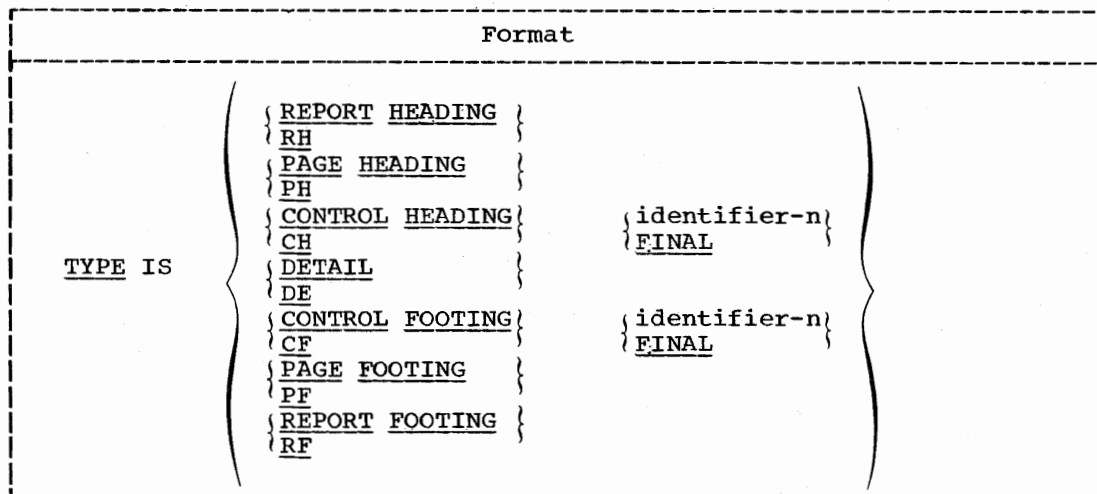
The TYPE clause in a particular report group entry indicates the point in time at which this report group will be generated as output.

If the report group is described as TYPE DETAIL or DE, then a GENERATE statement in the Procedure Division directs the Report Writer to produce this report group. Each DETAIL report group must have a unique data-name at level-01 in a report.

If the report group is described as other than TYPE DETAIL or DE, the generation of this report group is an automatic feature of the Report Writer, as detailed in the following paragraphs.

The REPORT HEADING or RH entry indicates a report group that is produced only once at the beginning of a report during the execution of the first GENERATE statement. There can be only one report group of this type in a report. SOURCE clauses used in REPORT HEADING report groups refer to the values of data items at the time the first GENERATE statement is executed.

The PAGE HEADING or PH entry indicates a report group that is produced at the beginning of each page according to PAGE condition rules as specified below. There can be only one report group of this type in a report.

The CONTROL HEADING or CH entry indicates a report group that is produced at the beginning of a control group for a designated identifier, or, in the case of FINAL, is produced once before the first control group during the execution of the first GENERATE statement. There can be only one report group of this type for each identifier and for the FINAL entry specified in a report. In order to produce any CONTROL HEADING report groups, a control break must occur. SOURCE clauses used in CONTROL HEADING FINAL report groups refer to the values of the items at the time the first GENERATE statement is executed.

The CONTROL FOOTING or CF entry indicates a report group that is produced at the end of a control group for a designated identifier or is

produced once at the termination of a report ending a FINAL control group. There can be only one report group of this type for each identifier and for the FINAL entry specified in a report. In order to produce any CONTROL FOOTING report groups, a control break must occur. SOURCE clauses used in CONTROL FOOTING FINAL report groups refer to the values of the items at the time the TERMINATE statement is executed.

The PAGE FOOTING or PF entry indicates a report group that is produced at the bottom of each page according to PAGE condition rules as specified below. There can be only one report group of this type in a report.

The REPORT FOOTING or RF entry indicates a report group that is produced only at the termination of a report. There can be only one report group of this type in a report. SOURCE clauses used in TYPE REPORT FOOTING report groups refer to the value of items at the time the TERMINATE statement is executed.

Identifier-n, as well as FINAL, must be one of the identifiers described in the CONTROL clause in the Report Description entry.

A FINAL type control break may be designated only once for CONTROL HEADING or CONTROL FOOTING entries within a particular report description.

Nothing precedes a REPORT HEADING entry and nothing follows a REPORT FOOTING entry within a report.

The HEADING or FOOTING report groups occur in the following Report Writer sequence if all exist for a given report:

```
REPORT HEADING (one occurrence only)
PAGE HEADING
    .
    .
    .
CONTROL HEADING
DETAIL
CONTROL FOOTING
    .
    .
    .
PAGE FOOTING
REPORT FOOTING (one occurrence only)
```

CONTROL HEADING report groups are presented in the following hierarchical arrangement:

```
Final Control Heading (one occurrence only)
Major Control Heading
    .
    .
    .
Minor Control Heading
```

CONTROL FOOTING report groups are presented in the following hierarchical arrangement:

```
Minor Control Footing
    .
    .
    .
Major Control Footing
Final Control Footing (one occurrence only)
```

CONTROL HEADING report groups appear with the current values of any indicated SOURCE data items before the DETAIL report groups of the CONTROL group are produced. CONTROL FOOTING report groups appear with the previous values of any indicated SOURCE data items specified in the CONTROL clause, just after the DETAIL report groups of that CONTROL group have been produced.

The USE procedures specified for a CONTROL FOOTING report group that refer to:

- source data items that are specified in the CONTROL clause affect the previous value of the items

- source data items that are not specified in the CONTROLS clause affect the current value of the items

These report groups appear whenever a control break occurs. LINE NUMBER determines the absolute or relative position of the CONTROL report groups exclusive of the other HEADING and FOOTING report groups.

USAGE Clause

DISPLAY is the only option that may be specified for group and elementary items in a Report Group Description entry (see "USAGE Clause").

COLUMN Clause

The COLUMN clause indicates the absolute column number on the printed page of the high-order (leftmost) character of an elementary item.

```
-------------------------------------------------------------------
|                              Format                             |
|-----------------------------------------------------------------|
|                                                                 |
|   COLUMN NUMBER IS integer                                      |
|                                                                 |
-------------------------------------------------------------------
```

The COLUMN clause indicates that the leftmost character of the elementary item is placed in the position specified by integer. If the column number is not indicated, the elementary item, though included in the description of the report group, is suppressed when the report group is produced at object time.

Integer must be a positive integer.

The COLUMN number clause is given at the elementary level within a report group even if the elementary level is a single level-01 entry, which alone constitutes the report group.

Within a report group and a particular LINE NUMBER specification, COLUMN number entries need not be indicated from left to right.

GROUP INDICATE Clause

The GROUP INDICATE clause specifies that this elementary item is to be produced only on the first occurrence of the item after any control or page break.

```
r--------------------------------------------------------------------¬
|                              Format                                |
|-------------------------------------------------------------------|
|                                                                   |
|  GROUP INDICATE                                                   |
|                                                                   |
L--------------------------------------------------------------------J
```

The GROUP INDICATE clause must be specified only at the elementary item level within a DETAIL report group.

An elementary item is not only group indicated in the first DETAIL report group containing the item after a control break, but is also group indicated in the first DETAIL report group containing the item on a new page, even though a control break did not occur.

JUSTIFIED Clause

The JUSTIFIED clause is applicable in report group description entries as described in "Data Division."

PICTURE Clause

The PICTURE clause is applicable in Report Group Description entries as described in "Data Division."

RESET Clause

The RESET clause indicates the CONTROL identifier that causes the SUM counter in the elementary item entry to be reset to zero on a CONTROL break.

```
r--------------------------------------------------------------------¬
|                              Format                                |
|-------------------------------------------------------------------|
|                                                                   |
|                       (identifier)                                |
|  RESET   ON          <           >                                |
|                       (FINAL     )                                |
|                                                                   |
L--------------------------------------------------------------------J
```

After presentation of the CONTROL FOOTING report group, the counters associated with the report group are reset automatically to zero,

unlessan explicit RESET clause is given specifying reset based on a higher level control than the associated control for the report group.

The RESET clause may be used for progressive totaling of identifiers where subtotals of identifiers may be desired without automatic resetting upon producing the report group.

Identifier must be one of the identifiers described in the CONTROL clause in the Report Description entry (RD). Identifier must be a higher level CONTROL identifier than the CONTROL identifier associated with the CONTROL FOOTING report group in which the SUM and RESET clauses appear.

The RESET clause may be used only in conjunction with a SUM clause.

BLANK WHEN ZERO Clause

The BLANK WHEN ZERO clause is applicable here as discussed in "Data Division."

SOURCE, SUM, or VALUE Clause

The SOURCE, SUM, or VALUE clause defines the purpose of this elementary item within the report group.

```
-----------------------------------------------------------------
|                            Format                             |
|---------------------------------------------------------------|
|                                                               |
|                         (  TALLY      )                       |
|        SOURCE IS        {             }                       |
|                         ( identifier-1)                       |
|                                                               |
|---------------------------------------------------------------|
|                                                               |
|             ( TALLY       ) [ ( TALLY      ) ]                |
|        SUM  {             } [ {            } ] ...[UPON data-name] |
|             ( identifier-2) [ ( identifier-3) ]              |
|                                                               |
|---------------------------------------------------------------|
|                                                               |
|        VALUE IS literal-1                                     |
|                                                               |
-----------------------------------------------------------------
```

SOURCE: The SOURCE clause indicates a data item that is to be used as the source for this report item. The item is presented according to the PICTURE clause and the COLUMN clause in this elementary item entry.

The SOURCE clause has two functions:

1. to specify a data item that is to be printed

2. to specify a data item that is to be summed in a CONTROL FOOTING report group (see SUM clause)

TALLY may be used in place of identifier-1 in a SOURCE clause.

SUM: The SUM clause is used to cause automatic summation of data and
may appear only in an elementary item entry of a CONTROL FOOTING report
group. The presence of a SUM clause defines a SUM counter. If a SUM
counter is to be referred to by a Procedure Division statement or Report
Section entry, a data-name clause must be specified with the SUM clause
entry. The data-name then represents the summation counter generated by
the Report Writer to total the operands specified immediately following
SUM. If reference is never made to a summation counter, the counter
need not be named explicitly by a data-name entry.

Whether the elementary item entry that contains a SUM clause names
the summation counter or not, the PICTURE clause must be specified for
each SUM counter. Editing characters or editing clauses may be included
in the description of a SUM counter. Editing of a SUM counter occurs
only upon presentation of that SUM counter. At all other times, the SUM
counter is treated as a numeric data item. The SUM counter must be
large enough to accommodate the summed quantity without truncation of
integral digits.

An operand of a SUM clause must be an elementary numeric data item
that appears in the File, Working-Storage, or Linkage Section, or is the
name of a SUM counter. A SUM counter that is an operand of SUM clause
must be defined in the same CONTROL FOOTING report group that contains
this SUM clause or in a CONTROL FOOTING report group that is at a lower
level in the control hierarchy of this report.

A SUM counter is incremented by its operands in the following manner:

• An operand that is an elementary numeric data item appearing in the
  File, Working-Storage, or Linkage Section is added to the SUM
  counter upon the generation of a DETAIL report group that contains
  this operand as a SOURCE data item; even if the operand appears in
  more than one SOURCE clause of the DETAIL report group, it is added
  only once to the SUM counter. The operands must appear exactly as
  they are in the SOURCE clauses with regard to qualification,
  subscripting, and indexing. If the SUM clause contains the UPON
  option, the operand does not have to appear in a SOURCE clause of
  the DETAIL report group.

• An operand that is a SUM counter and is defined in a CONTROL FOOTING
  that is at any lower level in the control hierarchy of this report
  is summed before presentation of the CONTROL FOOTING in which it is
  defined. This counter updating is commonly called rolling counters
  forward.

• An operand that is a SUM counter and is defined in the same CONTROL
  FOOTING as this SUM clause, is summed before presentation of this
  CONTROL FOOTING. This counter updating is commonly called
  cross-footing. SUM counter operands are added to their respective
  SUM counters in the order in which they physically appear in the
  CONTROL FOOTING report group description, i.e., left to right within
  an elementary item entry and down the elementary item entries.

The UPON data-name option is required to obtain selective summation for
a particular data item that is named as a SOURCE item in two or more
DETAIL report groups. Identifier-2 and identifier-3 must be SOURCE data
items in data-name. However, this compiler does not require that
identifier-2, identifier-3, etc., be SOURCE data items in data-name.
Data-name must be the name of a DETAIL report group.

The following is the chronology of summing events.

1. Cross-footing and counter rolling.

2. Execution of the USE BEFORE REPORTING procedure.

3. Presentation of the control footing if it is not a null group.

4. SUM counter resetting unless an explicit RESET clause appears in the entry that defines the SUM counter.

TALLY may be used in place of an identifier in the SUM clause. It is treated as an elementary numeric data item.

VALUE: The VALUE clause causes the report data item to assume the specified value each time its report group is presented only if the elementary item entry does not contain a GROUP INDICATE clause. If the GROUP INDICATE clause is present and a given object time condition exists, the item will not assume the specified value (see GROUP INDICATE rules).

PROCEDURE DIVISION CONSIDERATIONS

To produce a report, the INITIATE, GENERATE, and TERMINATE statements must be specified in the Procedure Division. In addition, a USE BEFORE REPORTING declarative may be written in a Declarative Section of the Procedure Division. This option allows the programmer to manipulate or alter data immediately before it is printed.

GENERATE Statement

The GENERATE statement is used to produce a report.

```
---------------------------------------------------------------------
|                             Format                                  |
|--------------------------------------------------------------------|
|                                                                     |
|   GENERATE identifier                                               |
|                                                                     |
---------------------------------------------------------------------
```
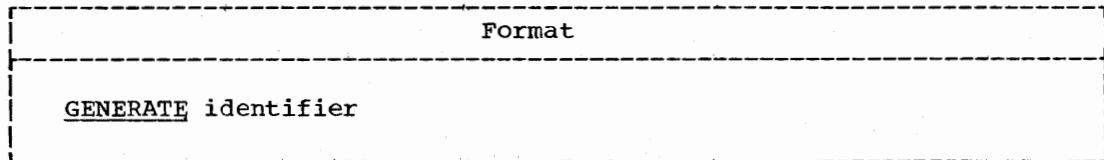
Identifier is the name of either a DETAIL report group or an RD entry.

If identifier is the name of a DETAIL report group, the GENERATE statement does all the automatic operations within a Report Writer program and produces an actual output detail report group on the output medium. This is called detail reporting.

If identifier is the name of an RD entry, the GENERATE statement does all of the automatic operations of the Report Writer except producing any detail report group associated with the report. This is called summary reporting.

In summary reporting, SUM counters are algebraically incremented in the same manner as for detail reporting. If more than one DETAIL report group is specified in a report, SUM counters are algebraically incremented as though consecutive GENERATE statements were issued for all the DETAIL report groups of the report. This consecutive summing takes place in the order of the physical appearance of the DETAIL report group descriptions. Even if there is more than one DETAIL report group within a report, only one test for control break is made for each GENERATE report-name. This test is made by the Report Writer prior to the summary reporting. After initiating a report and before terminating the same report, both detail reporting and summary reporting may be performed.

INITIATE Statement

A GENERATE statement, implicitly in both detail and summary
reporting, produces the following automatic operations (if defined):

1. Steps and tests the LINE COUNTER and/or PAGE COUNTER to produce
    appropriate PAGE FOOTING and/or PAGE HEADING report groups, after a
    line is printed.

2. Recognizes any specified control breaks to produce appropriate
    CONTROL FOOTING and/or CONTROL HEADING report groups.

3. Accumulates into the SUM counters all specified identifier(s).
    Resets the SUM counters.

4. Executes any specified routines defined by a USE statement before
    generation of the associated report group(s).

During the execution of the first GENERATE statement, the following
report groups associated with the report (if specified) are produced in
the order:

1. REPORT HEADING report group

2. PAGE HEADING report group

3. All CONTROL HEADING report groups in the order FINAL, major to
    minor

4. The DETAIL report group if specified in the GENERATE statement.

If a control break is recognized at the time of the execution of a
GENERATE statement (other than the first that is executed for a report),
all CONTROL FOOTING report groups specified for the report are produced
from the minor report group, up to and including the report group speci-
fied for the identifier which caused the control break. Then, the
CONTROL HEADING report group(s) specified for the report are produced,
starting with the report group specified for the identifier that caused
the control break, and continuing down to and ending with the minor
report group. Then, the DETAIL report group specified in the GENERATE
statement is produced.

Data is moved to the data item in the Report Group Description entry
of the Report Section and is edited under control of the Report Writer
according to the same rules for movement and editing as described for
the MOVE statement (see "Procedure Division").

INITIATE Statement

The INITIATE statement begins the processing of a report.

```
r----------------------------------------------------------------------
|                               Format                                 |
|----------------------------------------------------------------------|
|                                                                      |
|     INITIATE report-name-1 [report-name-2] ...                       |
|                                                                      |
L----------------------------------------------------------------------J
```

Each report-name must be defined by a Report Description entry in the
Report Section of the Data Division.

The INITIATE statement resets all data-name entries that contain SUM clauses associated with the report; the Report Writer controls for all the TYPE report groups that are associated with this report are set up in their respective order.

The PAGE-COUNTER register, if specified, is set to 1 (one) during the execution of the INITIATE statement. If a starting value other than 1 is desired, the programmer may reset this PAGE-COUNTER following the INITIATE statement.

The LINE-COUNTER register, if specified, is set to zero during the execution of the INITIATE statement.

The PRINT-SWITCH register is set to zero during the execution of the INITIATE statement.

The INITIATE statement does not open the file with which the report is associated; an OPEN statement for the file must be given by the user. The INITIATE statement performs Report Writer functions for individually described reports analogous to the input and/or output functions that the OPEN statement performs for individually described files.

A second INITIATE statement for a particular report-name may not be executed unless a TERMINATE statement has been executed for that report-name subsequent to the first INITIATE statement.

TERMINATE Statement

The TERMINATE statement completes the processing of a report.

```
+-----------------------------------------------------------------------+
|                                Format                                 |
+-----------------------------------------------------------------------+
|                                                                       |
|    TERMINATE report-name-1 [report-name-2] ...                        |
|                                                                       |
+-----------------------------------------------------------------------+
```

Each report-name given in a TERMINATE statement must be defined by an RD entry in the Data Division.

The TERMINATE statement produces all the CONTROL FOOTING report groups associated with this report as though a control break had just occurred at the highest level, and completes the Report Writer functions for the named reports. The TERMINATE statement also produces the last REPORT FOOTING report group associated with this report.

Appropriate PAGE HEADING and/or PAGE FOOTING report groups are prepared in their respective order for the report description.

A second TERMINATE statement for a particular report may not be executed unless a second INITIATE statement has been executed for the report-name.

The TERMINATE statement does not close the file with which the report is associated; a CLOSE statement for the file must be given by the user. The TERMINATE statement performs Report Writer functions for individually described report programs analogous to the input/output functions that the CLOSE statement performs for individually described files.

USE BEFORE REPORTING Declarative

If, at object time, no GENERATE statement is executed for a report, the TERMINATE statement of the report will not produce any report groups and will not perform any SUM processing.

SOURCE clauses used in CONTROL FOOTING FINAL or REPORT FOOTING report groups refer to the values of the items during the execution of the TERMINATE statement.

USE Sentence

The USE sentence specifies Procedure Division statements that are executed just before a report group named in the Report Section of the Data Division is produced.

```
r--------------------------------------------------------------------1
|                             Format                                 |
|--------------------------------------------------------------------|
|                                                                    |
|   USE BEFORE REPORTING data-name.                                  |
|                                                                    |
L--------------------------------------------------------------------J
```

A USE sentence, when present, must immediately follow a section header in the declaratives portion of the Procedure Division and must be followed 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.

Data-name represents a report group named in the Report Section of the Data Division. A data-name must not appear in more than one USE sentence. Data-name must be qualified by the report-name if data-name is not unique.

No Report Writer statement (GENERATE, INITIATE, or TERMINATE) may be written in a procedural paragraph(s) following the USE sentence in the declaratives portion.

The USE sentence itself is never executed; rather it defines the conditions calling for the execution of the USE procedures.

The designated procedures are executed by the Report Writer just before the named report is produced, regardless of page or control break associations with report groups. The report group may be any type except DETAIL.

Within a USE procedure, there must not be any reference to any nondeclarative procedures. Conversely, in the nondeclarative portion, there must be no reference to procedure names that appear in the Declaratives Section, except that PERFORM statements may refer to a USE procedure or to the procedures associated with the USE procedure.

When the user wishes to suppress the printing of the specified report groups, the statement

    MOVE 1 TO PRINT-SWITCH

is used in the USE BEFORE REPORTING declarative section. When this statement is encountered, only the specified report group is not printed; the statement must be written for each report group whose printing is to be suppressed.

Use of PRINT-SWITCH to suppress the printing of a report group
implies that:

1. nothing is printed

2. the LINE-COUNTER is not altered

3. the function of the NEXT GROUP clause, if one appears in the report
   group description, is nullified

SPECIAL REGISTERS: PAGE-COUNTER AND LINE-COUNTER

The fixed data-names, PAGE-COUNTER and LINE-COUNTER, are numeric
counters automatically generated by the Report Writer based on the
presence of specific entries; they do not require data description
clauses. The description of these two counters is included here in
order to explain their resultant effect on the overall report format.

PAGE-COUNTER

A PAGE-COUNTER is a counter generated by the Report Writer to be used
as a source data item in order to present the page number on a report
line. A PAGE-COUNTER is generated for a report by the Report Writer if
a PAGE-LIMIT clause is specified in the RD entry of the report. The
numeric counter is a 3-byte COMPUTATIONAL-3 item that is presented
according to the PICTURE clause associated with the elementary item
whose SOURCE is PAGE-COUNTER.

If more than one PAGE-COUNTER is given as a SOURCE data item within a
given report, the number of numeric characters indicated by the PICTURE
clauses must be identical. If more than one PAGE-COUNTER exists in the
program, the user must qualify PAGE-COUNTER by the report name.

PAGE-COUNTER may be referred to in Report Section entries and in
Procedure Division statements. After an INITIATE statement,
PAGE-COUNTER contains one; if a starting value for PAGE-COUNTER other
than one is desired, the programmer may change the contents of the
PAGE-COUNTER by a Procedure Division statement after an INITIATE
statement has been executed. PAGE-COUNTER is automatically incremented
by one each time a page break is recognized by the Report Writer, after
the production of any PAGE FOOTING report group but before production of
any PAGE HEADING report group.

LINE-COUNTER

A LINE-COUNTER is a counter used by the Report Writer to determine
when a PAGE HEADING and/or a PAGE FOOTING report group is to be
presented. One line counter is supplied for each report with a PAGE
LIMIT(S) clause written in the Report Description entry (RD). The
numeric counter is a 3-byte COMPUTATIONAL-3 item that is presented
according to the PICTURE clause associated with the elementary item
whose SOURCE is LINE-COUNTER.

LINE-COUNTER may be referred to in Report Section entries and in
Procedure Division statements. If more than one Report Description
entry (RD) exists in the Report Section, the user must qualify
LINE-COUNTER by the report-name. LINE-COUNTER is automatically tested
and incremented by the Report Writer based on control specifications in
the PAGE LIMIT(S) clause and values specified in the LINE NUMBER and

NEXT GROUP clauses. After an INITIATE statement, LINE-COUNTER contains zero. Changing the value of LINE-COUNTER by Procedure Division statements may cause page format control to become unpredictable in the Report Writer.

The value of LINE-COUNTER during any Procedure Division test statement represents the number of the last line printed by the previously generated report group or represents the number of the last line skipped to by a previous NEXT GROUP specification.

In a USE BEFORE REPORTING, if no lines have been printed or skipped on the current page, LINE-COUNTER will contain zero. In all other cases, LINE-COUNTER represents the last line printed or skipped.

SAMPLE REPORT WRITER PROGRAM

Figure 16 illustrates a Report Writer source program. The records used in the report (i.e., input data) are shown after the STOP RUN card in the program. Using the first record as an example, the data fields are arranged in the following card format:
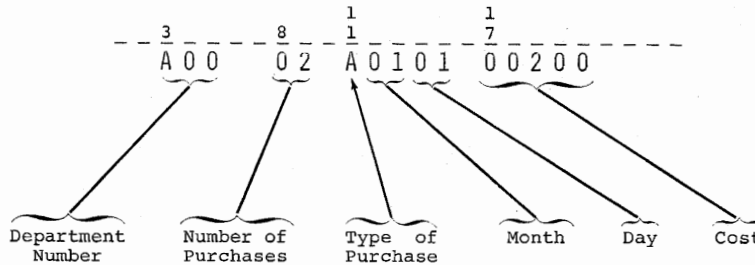
The decimal point in the cost field is assumed to be two places from the right.

```
|000005    IDENTIFICATION DIVISION.
|000010    PROGRAM-ID. ACME.
|000015    REMARKS. THE REPORT WAS PRODUCED BY THE REPORT WRITER.
|000020    ENVIRONMENT DIVISION.
|000025    CONFIGURATION SECTION.
|000030    SOURCE-COMPUTER.   IBM-360-F50.
|000035    OBJECT-COMPUTER.   IBM-360-F50.
|000040    INPUT-OUTPUT SECTION.
|000045    FILE-CONTROL.
|000050        SELECT INFILE ASSIGN TO SYS000-UT-2400-S.
|000055        SELECT REPORT-FILE ASSIGN TO SYS001-UT-2400-S.
|000060    DATA DIVISION.
|000065    FILE SECTION.
|000070    FD   INFILE
|000075        LABEL RECORDS ARE OMITTED
|000080        DATA RECORD IS INPUT-RECORD.
|000085    01   INPUT-RECORD.
|000090        02   FILLER          PICTURE AA.
|000095        02   DEPT            PICTURE XXX.
|000100        02   FILLER          PICTURE AA.
|000105        02   NO-PURCHASES    PICTURE 99.
|000110        02   FILLER          PICTURE A.
|000115        02   TYPE-PURCHASE   PICTURE A.
|000120        02   MONTH           PICTURE 99.
|000125        02   DAY             PICTURE 99.
|000130        02   FILLER          PICTURE A.
|000135        02   COST            PICTURE 999V99.
|000140        02   FILLER          PICTURE X(59).
|000145    FD   REPORT-FILE
|000150        LABEL RECORDS ARE STANDARD
|000151        RECORD CONTAINS 121 CHARACTERS
|000155        REPORT IS EXPENSE-REPORT.
|000160    WORKING-STORAGE SECTION.
|000165    77   SAVED-MONTH         PICTURE 99 VALUE IS 0.
|000175    77   CONTINUED           PICTURE X(11) VALUE IS SPACE.
```

Figure 16. Sample Program Using the Report Writer Feature (Part 1 of 4)

```
000180  01   MONTH-TABLE-1.
000185       02   RECORD-MONTH.
000190            03   FILLER   PICTURE A(9) VALUE IS "JANUARY  ".
000195            03   FILLER   PICTURE A(9) VALUE IS "FEBRUARY ".
000200            03   FILLER   PICTURE A(9) VALUE IS "MARCH    ".
000205            03   FILLER   PICTURE A(9) VALUE IS "APRIL    ".
000210            03   FILLER   PICTURE A(9) VALUE IS "MAY      ".
000215            03   FILLER   PICTURE A(9) VALUE IS "JUNE     ".
000220            03   FILLER   PICTURE A(9) VALUE IS "JULY     ".
000225            03   FILLER   PICTURE A(9) VALUE IS "AUGUST   ".
000230            03   FILLER   PICTURE A(9) VALUE IS "SEPTEMBER".
000235            03   FILLER   PICTURE A(9) VALUE IS "OCTOBER  ".
000240            03   FILLER   PICTURE A(9) VALUE IS "NOVEMBER ".
000245            03   FILLER   PICTURE A(9) VALUE IS "DECEMBER ".
000250       02   RECORD-AREA REDEFINES RECORD-MONTH.
000255            03   MONTHNAME    PICTURE A(9) OCCURS 12 TIMES.
000260  REPORT SECTION.
000265  RD   EXPENSE-REPORT
000270       CONTROLS ARE FINAL MONTH DAY
000275       PAGE LIMIT IS 59 LINES
000280       HEADING 1
000285       FIRST DETAIL 9
000290       LAST DETAIL 48
000295       FOOTING 52.
000300  01   TYPE IS REPORT HEADING.
000305       02   LINE NUMBER IS 1
000310            COLUMN NUMBER IS 27
000315            PICTURE IS A(26)
000320            VALUE IS "ACME MANUFACTURING COMPANY".
000325       02   LINE NUMBER IS 3
000330            COLUMN NUMBER IS 26
000335            PICTURE IS A(29)
000340            VALUE IS "QUARTERLY EXPENDITURES REPORT".
000345  01   PAGE-HEAD
000350       TYPE IS PAGE HEADING.
000355       02   LINE NUMBER IS 5.
000360            03   COLUMN IS 30
000365                 PICTURE IS A(9)
000370                 SOURCE IS MONTHNAME OF RECORD-AREA (MONTH).
000375            03   COLUMN IS 39
000380                 PICTURE IS A(12)
000385                 VALUE IS "EXPENDITURES".
000390            03   COLUMN IS 52
000395                 PICTURE IS X(11)
000400                 SOURCE IS CONTINUED.
000405       02   LINE IS 7.
000410            03   COLUMN IS 2
000415                 PICTURE IS X(35)
000420                 VALUE IS "MONTH      DAY   DEPT  NO-PURCHASES".
000425            03   COLUMN IS 40
000430                 PICTURE IS X(33)
000435                 VALUE IS "TYPE   COST CUMULATIVE-COST".
```

Figure 16. Sample Program Using the Report Writer Feature (Part 2 of 4)

```
|000440  01    DETAIL-LINE TYPE IS DETAIL LINE NUMBER IS PLUS 1.
|000445      02    COLUMN IS 2 GROUP INDICATE PICTURE IS A(9)
|000450            SOURCE IS MONTHNAME OF RECORD-AREA (MONTH).
|000455      02    COLUMN IS 13 GROUP INDICATE PICTURE IS 99
|000460            SOURCE IS DAY.
|000465      02    COLUMN IS 19 PICTURE IS XXX SOURCE IS DEPT.
|000470      02    COLUMN IS 31 PICTURE IS Z9 SOURCE IS NO-PURCHASES.
|000475      02    COLUMN IS 42 PICTURE IS A SOURCE IS TYPE-PURCHASE.
|000480      02    COLUMN IS 50 PICTURE IS ZZ9.99 SOURCE IS COST.
|000485  01    TYPE IS CONTROL FOOTING DAY.
|000490      02    LINE NUMBER IS PLUS 2.
|000495          03    COLUMN 2 PICTURE X(22)
|000500                VALUE "PURCHASES AND COST FOR".
|000505          03    COLUMN 24   PICTURE Z9 SOURCE SAVED-MONTH.
|000510          03    COLUMN 26   PICTURE X VALUE "-".
|000515          03    COLUMN 27   PICTURE 99 SOURCE DAY.
|000520          03    COLUMN 30   PICTURE ZZ9 SUM NO-PURCHASES.
|000525          03    MIN
|000530                COLUMN 49   PICTURE $$$9.99 SUM COST.
|000535          03    COLUMN 65   PICTURE $$$$9.99 SUM COST
|000540                RESET ON FINAL.
|000545      02    LINE PLUS 1 COLUMN 2 PICTURE X(71)
|000550            VALUE ALL "*".
|000555  01    TYPE CONTROL FOOTING MONTH
|000560      LINE PLUS 1 NEXT GROUP NEXT PAGE.
|000565      02    COLUMN 16        PICTURE A(14) VALUE "TOTAL COST FOR".
|000570      02    COLUMN 31        PICTURE A(9)
|000575            SOURCE MONTHNAME OF RECORD-AREA (MONTH).
|000580      02    COLUMN 43        PICTURE AAA VALUE "WAS".
|000585      02    INT
|000590            COLUMN 48        PICTURE $$$9.99 SUM MIN.
|000595  01    TYPE CONTROL FOOTING FINAL LINE NEXT PAGE.
|000600      02    COLUMN 16        PICTURE A(26)
|000605            VALUE "TOTAL COST FOR QUARTER WAS".
|000610      02    COLUMN 45        PICTURE $$$$9.99 SUM INT.
|000615  01    TYPE PAGE FOOTING LINE 57.
|000620      02    COLUMN 59        PICTURE X(12) VALUE "REPORT-PAGE-".
|000625      02    COLUMN 71        PICTURE 99 SOURCE PAGE-COUNTER.
|000630  01    TYPE REPORT FOOTING
|000635      LINE PLUS 1 COLUMN 32 PICTURE A(13)
|000640      VALUE "END OF REPORT".
|000645  PROCEDURE DIVISION.
|000650  DECLARATIVES.
|000655  PAGE-HEAD-RTN SECTION.
|000660      USE BEFORE REPORTING PAGE-HEAD.
|000665  PAGE-HEAD-RTN-SWITCH.
|000670      GO TO PAGE-HEAD-RTN-TEST.
|000675  PAGE-HEAD-RTN-TEST.
|000680      IF MONTH = SAVED-MONTH MOVE "(CONTINUED)" TO CONTINUED
|000685          ELSE MOVE SPACES TO CONTINUED
|000690              MOVE MONTH TO SAVED-MONTH.
|000695      GO TO PAGE-HEAD-RTN-EXIT.
|000697  PAGE-HEAD-RTN-ALTER.
|000698      ALTER PAGE-HEAD-RTN-SWITCH TO PAGE-HEAD-RTN-SUPPRESS.
|000700  PAGE-HEAD-RTN-SUPPRESS.
|000705      MOVE 1 TO PRINT-SWITCH.
|000710  PAGE-HEAD-RTN-EXIT.
|000715      EXIT.
|000720  END DECLARATIVES.
```

Figure 16.   Sample Program Using the Report Writer Feature (Part 3 of 4)

```
|000725    OPEN-FILES.   OPEN INPUT INFILE OUTPUT REPORT-FILE.
|000730        INITIATE EXPENSE-REPORT.
|000735    READATA.
|000740        READ INFILE AT END GO TO COMPLETE.
|000745        GENERATE DETAIL-LINE.
|000760        GO TO READATA.
|000765    COMPLETE.
|000770        PERFORM PAGE-HEAD-RTN-ALTER.
|000780        TERMINATE EXPENSE-REPORT.
|000785        CLOSE INFILE REPORT-FILE.
|000790        STOP RUN.
|               .
|               .
|               .
|   A00   02 A0101 00200
|   A02   01 A0101 00100
|   A02   01 A0101 01600
|   A01   02 B0102 00200
|   A04   10 A0102 01000
|   A01   20 D0108 03840
|               .
|               .
|               .
|   A01   06 C0329 04800
|   A03   20 E0331 0600
|   A03   10 G0331 05000
```

Figure 16.   Sample Program Using the Report Writer Feature (Part 4 of 4)

Key Relating Report to Report Writer Source Program

In the key, the numbers enclosed in circles (for example, (1)) relate the explanation below to the corresponding output line in Figure 17.

The six-digit numbers (for example, 000615) show the source statement from the program illustrated in Figure 16.

(1)   is the REPORT HEADING resulting from source lines 000300-000340.

(2)   is the PAGE HEADING resulting from source lines 000345-000435.

(3)   is the DETAIL line resulting from source lines 000440-000480 (note
       that since it is the first detail line after a control break, the
       fields defined with the GROUP INDICATE clause, lines
       000445-000460, appear).

(4)   is a DETAIL line resulting from the same source lines as   (3)
       In this case, however, the fields described as GROUP INDICATE do
       not appear (since the control break did not immediately precede
       the detail line).

(5)   is the CONTROL FOOTING (for DAY) resulting from source lines
       000485-000550.

(6)   is the PAGE FOOTING resulting from source lines 000615-000625.

⑦ is the CONTROL FOOTING (for MONTH) resulting from source lines 000555-000575.

⑧ is the CONTROL FOOTING (for FINAL) resulting from source lines 000595-000610.

⑨ is the REPORTING FOOTING resulting from source lines 000630-000640.

Lines 000650-000715 of the example illustrate a use of USE BEFORE REPORTING. The effect of the source is that each time a new page is started, a test is made to determine whether the new page is being started because a change in MONTH has been recognized (the definition for the control footing for MONTH specifies NEXT GROUP NEXT PAGE) or because the physical limits of the page were exhausted. If a change in MONTH has been recognized, spaces are moved to the PAGE HEADING; if the physical limits of the page are exhausted, "(CONTINUED)" is moved to the PAGE HEADING.

```
①————————————————————————————ACME MANUFACTURING COMPANY

                            QUARTERLY EXPENDITURES REPORT

②————————————————————————————JANUARY  EXPENDITURES

   MONTH      DAY    DEPT   NO-PURCHASES    TYPE       COST   CUMULATIVE-COST

③——JANUARY    01     A00          2          A        2.00
                    ┌A02          1          A        1.00
④                   A02          2          C       16.00

 ┌PURCHASES AND COST FOR 1-01   5                   $19.00           $19.00
⑤ ****************************************************************************
   JANUARY    02     A01          2          B        2.00
                     A04         10          A       10.00
                     A04         10          C       80.00

   PURCHASES AND COST FOR 1-02  22                   $92.00          $111.00
   ****************************************************************************
   JANUARY    05     A01          2          B        2.00

   PURCHASES AND COST FOR 1-05   2                    $2.00          $113.00
   ****************************************************************************
   JANUARY    08     A01         10          A       10.00
                    ¦A01          8          B       12.48
                     A01         20          D       38.40

   PURCHASES AND COST FOR 1-08  38                   $60.88          $173.88
   ****************************************************************************
   JANUARY    13     A00          4          B        6.24
                     A00          1          C        8.00

   PURCHASES AND COST FOR 1-13   5                   $14.24          $188.12
   ****************************************************************************
   JANUARY    15     A00         10          D       19.20
                     A02          1          C        8.00

   PURCHASES AND COST FOR 1-15  11                   $27.20          $215.32
   ****************************************************************************
   JANUARY    21     A03         10          E       30.00
                     A03         10          F       25.00
                     A03         10          G       50.00

   PURCHASES AND COST FOR 1-21  30                  $105.00          $320.32
   ****************************************************************************
   JANUARY    23     A00          5          A        5.00

   PURCHASES AND COST FOR 1-23   5                    $5.00          $325.32
   ****************************************************************************




⑥————————————————————————————————————————————————————————REPORT-PAGE-01
```

Figure 17.   Report Produced by Report Writer Feature (Part 1 of 5)

```
 ②─────────────────────────────────────────JANUARY  EXPENDITURES (CONTINUED)

    MONTH      DAY     DEPT   NO-PURCHASES    TYPE       COST   CUMULATIVE-COST

 ③──JANUARY    26     A04         5            A        5.00
                     ─A04         5            B        7.80
 ④
   ─PURCHASES  AND  COST FOR  1-26  10                 $12.80           $338.12
 ⑤─*********************************************************************************
    JANUARY    27     A00         6            B        9.36
                      A00        15            C      120.00

    PURCHASES  AND  COST FOR  1-27  21                $129.36           $467.48
    *********************************************************************************
    JANUARY    30     A00         2            B        3.12
                      A02        10            A       10.00
                      A02         1            C        8.00
                      A04        15            B       23.40
                      A04        10            C       80.00

    PURCHASES  AND  COST FOR  1-30  38                $124.52           $592.00
    *********************************************************************************
    JANUARY    31     A00         1            A        1.00
                      A04         6            A        6.00

    PURCHASES  AND  COST FOR  1-31   7                 $7.00           $599.00
    *********************************************************************************
 ⑦────────────TOTAL COST FOR JANUARY     WAS   $599.00
```

```
 ⑥─────────────────────────────────────────────────────────────REPORT-PAGE-02
```

Figure 17.   Report Produced by Report Writer Feature (Part 2 of 5)

② —————————————————————————FEBRUARY EXPENDITURES

```
 MONTH       DAY    DEPT   NO-PURCHASES    TYPE        COST  CUMULATIVE-COST
③──FEBRUARY   15    A02        10           A         10.00
                    A02         2           B          3.12
                   ─A02         1           C          8.00
                    A03        15           G         75.00
④                   A04         5           B          7.80
                    A05         8           A          8.00
                    A05         5           C         40.00

⑤─PURCHASES AND COST FOR 2-15  46                   $151.92         $750.92
 ***********************************************************************
  FEBRUARY   16    A02         2           C         16.00
                   A06        10           A         10.00
                   A07        10           A         10.00
                   A07        10           F         25.00

 PURCHASES AND COST FOR 2-16  32                    $61.00         $811.92
 ***********************************************************************
  FEBRUARY   17    A07        10           E         30.00
                   A07        10           G         50.00

 PURCHASES AND COST FOR 2-17  20                    $80.00         $891.92
 ***********************************************************************
  FEBRUARY   21    A06        20           A         20.00
                   A06        20           B         31.20
                   A06        20           C        160.00
                   A06        20           D         38.40
                   A06        20           E         60.00
                   A06        20           F         50.00
                   A06        20           G        100.00

 PURCHASES AND COST FOR 2-21 140                   $459.60        $1351.52
 ***********************************************************************
  FEBRUARY   27    A01        21           D         40.32

 PURCHASES AND COST FOR 2-27  21                    $40.32        $1391.84
 ***********************************************************************
  FEBRUARY   28    A02         3           B          4.68
                   A02         5           C         40.00
                   A03        15           E         45.00

 PURCHASES AND COST FOR 2-28  23                    $89.68        $1481.52
 ***********************************************************************
⑦───────────────TOTAL COST FOR FEBRUARY    WAS  $882.52
```

⑥ ————————————————————————————————————REPORT-PAGE-03

Figure 17.   Report Produced by Report Writer Feature (Part 3 of 5)

```
(2)────────────────────────────────────MARCH     EXPENDITURES

     MONTH      DAY    DEPT   NO-PURCHASES    TYPE        COST   CUMULATIVE-COST
(3)───MARCH      01    A02          5          A         5.00
                      ─A02          1          C         8.00
(4)                    A03         25          G       125.00

    ─PURCHASES AND COST FOR 3-01  31                  $138.00         $1619.52
(5)─ ********************************************************************************
     MARCH      06    A02          5          A         5.00

     PURCHASES AND COST FOR 3-06   5                    $5.00         $1624.52
     ********************************************************************************
     MARCH      07    A02          5          A         5.00

     PURCHASES AND COST FOR 3-07   5                    $5.00         $1629.52
     ********************************************************************************
     MARCH      13    A02         10          A        10.00

     PURCHASES AND COST FOR 3-13  10                   $10.00         $1639.52
     ********************************************************************************
     MARCH      15    A01         21          A        21.00
                       A02          1          A         1.00
                       A03         15          F        37.50
                       A06          5          E        15.00
                       A06          5          F        12.50

     PURCHASES AND COST FOR 3-15  47                   $87.00         $1726.52
     ********************************************************************************
     MARCH      20    A03         15          E        45.00

     PURCHASES AND COST FOR 3-20  15                   $45.00         $1771.52
     ********************************************************************************
     MARCH      21    A02         15          A        15.00
                       A03         15          F        37.50

     PURCHASES AND COST FOR 3-21  30                   $52.50         $1824.02
     ********************************************************************************
     MARCH      23    A02          2          A         2.00

     PURCHASES AND COST FOR 3-23   2                    $2.00         $1826.02
     ********************************************************************************
     MARCH      25    A03         30          F        75.00

     PURCHASES AND COST FOR 3-25  30                   $75.00         $1901.02
     ********************************************************************************



(6)──────────────────────────────────────────────────────────REPORT-PAGE-04
```

Figure 17.   Report Produced by Report Writer Feature (Part 4 of 5)

(2) ─────────────────────────────── MARCH   EXPENDITURES (CONTINUED)

       MONTH   DAY   DEPT   NO-PURCHASES   TYPE   COST   CUMULATIVE-COST

(3) ── MARCH   26   A02   1   A   1.00

(5) ── PURCHASES AND COST FOR 3-26   1 $1.00 $1902.02
       ****
       MARCH   29   A01   6   C   48.00

       PURCHASES AND COST FOR 3-29   6 $48.00 $1950.02
       ****
       MARCH   31   A03   20   E   60.00

       PURCHASES AND COST FOR 3-31   20 $60.00 $2010.02
       ****
(7) ─────────────── TOTAL COST FOR MARCH   WAS $528.50

(6) ─── REPORT-PAGE-05

(8) ── TOTAL COST FOR QUARTER WAS $2010.02

(6) ─── REPORT-PAGE-06
(9) ─────────────────────── END OF REPORT

Figure 17.   Report Produced by Report Writer Feature (Part 5 of 5)

The Table Handling feature enables the programmer to process tables or lists of repeated data conveniently. A table may have up to three dimensions, i.e., three levels of subscripting or indexing can be handled. Such a case exists when a group item described with an OCCURS clause contains another group item with an OCCURS clause, which in turn contains an item with an OCCURS clause. To make reference to any element within such a table, each level must be subscripted or indexed.

SUBSCRIPTING

Subscripts are used only to refer to an individual element within a list or table of elements that have not been assigned individual data-names.

```
+-----------------------------------------------------------------------+
|                                Format                                 |
+-----------------------------------------------------------------------+
|                                                                       |
|   data-name (subscript[, subscript][, subscript])                     |
|                                                                       |
+-----------------------------------------------------------------------+
```

The subscript, or set of subscripts, that identifies the table element is enclosed in parentheses immediately following the space that terminates data-name, which is the name of the table element. When more than one subscript appears within a pair of parentheses, the subscripts must be separated by commas. A space must follow each comma, but no space may appear between the left parenthesis and the leftmost subscript or between the rightmost subscript and the right parenthesis. To identify an element in the table named SALARY by the set of subscripts YEAR, MONTH, and WEEK, the programmer would write:

SALARY (YEAR, MONTH, WEEK).

The subscript can be represented by a numeric literal that is a positive integer, by the special register TALLY, or by a data-name. Restrictions on the use of a data-name as a subscript are:

1. Data-name must be a numeric elementary item that represents a positive integer.

2. The name itself may be qualified, but not subscripted.

The subscript may contain a sign, but the lowest permissible subscript value is 1. Hence, the use of zero or a negative subscript is not permitted. The highest permissible subscript value in any particular case is the maximum number of occurrences of the item as specified in the OCCURS clause.

Qualification may be used in conjunction with subscripting, in which case OF or IN follows the data-name being subscripted.

```
┌─────────────────────────────────────────────────────────────────────────┐
│                                Format                                     │
├─────────────────────────────────────────────────────────────────────────┤
│                                                                           │
│             ⎧ OF ⎫               ⎧ OF ⎫                                   │
│  data-name ⎨    ⎬ data-name-1 [ ⎨    ⎬ data-name-2]...                    │
│             ⎩ IN ⎭               ⎩ IN ⎭                                   │
│                                                                           │
│         (subscript[, subscript][, subscript])                             │
│                                                                           │
└─────────────────────────────────────────────────────────────────────────┘
```

Note: Data-name is the item being subscripted, not data-name-1. That
is, in the statement SALARY OF EMPLOYEE-RECORD (YEAR, MONTH, WEEK), the
data item SALARY is subscripted by YEAR, MONTH, and WEEK.

INDEXING

   References can be made to individual elements within a table of
elements by specifying indexing for that reference.

   An index is assigned to a given level of a table by using an INDEXED
BY clause in the definition of the table. A name given in the INDEXED
BY clause is known as an index-name and is used to refer to the assigned
index. An index-name must be initialized by a SET statement before it
is used in a table reference. An index may be modified only by a SET,
SEARCH, or PERFORM statement. Data items described by the USAGE IS
INDEX clause permit storage of the values of index-names as data without
conversion. Such data items are called index data items.


```
┌─────────────────────────────────────────────────────────────────────────┐
│                                Format                                     │
├─────────────────────────────────────────────────────────────────────────┤
│                                                                           │
│                            ⎧ + ⎫                                          │
│  data-name (index-name [ ⎨   ⎬ integer]                                   │
│                            ⎩ - ⎭                                          │
│                                                                           │
│              ⎧ + ⎫                        ⎧ + ⎫                           │
│  [, index-name [ ⎨   ⎬ integer]][, index-name [ ⎨   ⎬ integer]])          │
│              ⎩ - ⎭                        ⎩ - ⎭                           │
│                                                                           │
└─────────────────────────────────────────────────────────────────────────┘
```


   Direct indexing is specified by using an index-name in the form of a
subscript. For example,

       ELEMENT (PRIME-INDEX)

   Relative indexing is specified when the terminal space of the
data-name is followed by a parenthesized group of items: the
index-name, followed by a space, followed by one of the operators +
or -, followed by another space, followed by an unsigned integral
numeric literal. For example,

       ELEMENT (PRIME-INDEX + 5)

   Qualification may be used in conjunction with indexing, in which case
OF or IN follows the data-name being indexed.

```
+-------------------------------------------------------------------------+
|                                 Format                                  |
+-------------------------------------------------------------------------+
|                                                                         |
|                  ( OF )                    ( OF )                        |
|     data-name   {    }  data-name-1  [ {    }  data-name-2]...           |
|                  ( IN )                    ( IN )                        |
|                                                                         |
|                       ( + )                             ( + )           |
|        (index-name  [ {   } integer][, index-name  [ {   } integer]]     |
|                       ( - )                             ( - )           |
|                                                                         |
|                            ( + )                                        |
|        [, index-name  [ {   } integer]])                                |
|                            ( - )                                        |
|                                                                         |
+-------------------------------------------------------------------------+
```

RESTRICTIONS ON INDEXING, SUBSCRIPTING, AND QUALIFICATION

Tables may have one, two, or three dimensions. Therefore, references to an element in a table may require up to three subscripts or indexes.

1. A data-name must not be subscripted or indexed when the data-name is itself being used as an index, subscript, or qualifier.

2. When qualification, subscripting, or indexing are required for a given data item, the indexes or subscripts are specified after all necessary qualification is given.

3. Subscripting and indexing must not be used together in a single reference.

4. Wherever subscripting is not permitted, indexing is not permitted.

5. The commas shown in the formats for indexes and subscripts are required.

EXAMPLE OF SUBSCRIPTING AND INDEXING

For a table with three levels of indexing, the following Data Division entries would result in a storage layout as shown in Figure 18.

```
01  PARTY-TABLE REDEFINES TABLE.
    02  PARTY-CODE OCCURS  3 TIMES INDEXED BY PARTY.
        03  AGE-CODE OCCURS 3 TIMES INDEXED BY AGE.
            04  M-F-INFO OCCURS 2 TIMES INDEXED BY M-F
                PICTURE 9(7)V9 USAGE DISPLAY.
```

PARTY-TABLE contains three levels of indexing. Reference to elementary items within PARTY-TABLE is made by use of a name that is subscripted or indexed. A typical Procedure Division statement might be:

MOVE M-F-INFO (PARTY, AGE, M-F) TO M-F-RECORD.

In order to use the Table Handling feature, the programmer must provide certain information in the Data Division and Procedure Division of the program.

```
r--------------------------------------------------------------------------------¬
|                                                          8 bytes        Byte|
|                                                                         No. |
|                                                        ~~~~~~~~~             |
|                                                      r------------------¬  0|
|                                                      |M-F-INFO (1, 1, 1)|    |
|                                   AGE-CODE (1, 1) {  |------------------|  8|
|                                                      |M-F-INFO (1, 1, 2)|    |
|                                                      |------------------| 16|
|                                                      |M-F-INFO (1, 2, 1)|    |
|                     /PARTY-CODE(1){ AGE-CODE (1, 2) {|------------------| 24|
|                     /                                |M-F-INFO (1, 2, 2)|    |
|                                                      |------------------| 32|
|                                                      |M-F-INFO (1, 3, 1)|    |
|                                   AGE-CODE (1, 3) {  |------------------| 40|
|                                                      |M-F-INFO (1, 3, 2)|    |
|                                                      |------------------| 48|
|                                                      |M-F-INFO (2, 1, 1)|    |
|                                   AGE-CODE (2, 1) {  |------------------| 56|
|                                                      |M-F-INFO (2, 1, 2)|    |
|                                                      |------------------| 64|
|                                                      |M-F-INFO (2, 2, 1)|    |
| PARTY-TABLE {  PARTY-CODE(2){  AGE-CODE (2, 2) {     |------------------| 72|
|                                                      |M-F-INFO (2, 2, 2)|    |
|                                                      |------------------| 80|
|                                                      |M-F-INFO (2, 3, 1)|    |
|                                   AGE-CODE (2, 3) {  |------------------| 88|
|                                                      |M-F-INFO (2, 3, 2)|    |
|                                                      |------------------| 96|
|                                                      |M-F-INFO (3, 1, 1)|    |
|                                   AGE-CODE (3, 1) {  |------------------|104|
|                                                      |M-F-INFO (3, 1, 2)|    |
|                                                      |------------------|112|
|                                                      |M-F-INFO (3, 2, 1)|    |
|                     \PARTY-CODE(3){ AGE-CODE (3, 2) {|------------------|120|
|                                                      |M-F-INFO (3, 2, 2)|    |
|                                                      |------------------|128|
|                                                      |M-F-INFO (3, 3, 1)|    |
|                                   AGE-CODE (3, 3) {  |------------------|136|
|                                                      |M-F-INFO (3, 3, 2)|    |
|                                                      L------------------┘    |
|                                                                             |
|              ~~~~~~~~~~~~     ~~~~~~~~~~~~        ~~~~~~~~~~~~               |
|              OCCURS 3 TIMES   OCCURS 3 TIMES      OCCURS 2 TIMES            |
L--------------------------------------------------------------------------------┘
```

Figure 18. Storage Layout for PARTY-TABLE

DATA DIVISION CONSIDERATIONS FOR TABLE HANDLING

The OCCURS and USAGE clauses are included as part of the record
description entries in a program utilizing the Table Handling feature.

OCCURS Clause

The OCCURS clause eliminates the need for separate entries for
repeated data, since it indicates the number of times a series of
records with identical format is repeated. It also supplies information
required for the application of subscripts or indexes.

The OCCURS clause has three formats.

```
+-----------------------------------------------------------------------+
|                            Format 1                                   |
+-----------------------------------------------------------------------+
|                                                                       |
| OCCURS integer-2 TIMES                                                |
|                                                                       |
|        ( ASCENDING  )                                                 |
|     [  {            } KEY IS data-name-2 [data-name-3] ... ] ...      |
|        ( DESCENDING )                                                 |
|                                                                       |
|           [INDEXED BY index-name-1[index-name-2] ... ]                |
|                                                                       |
+-----------------------------------------------------------------------+
```

```
+-----------------------------------------------------------------------+
|                            Format 2                                   |
+-----------------------------------------------------------------------+
|                                                                       |
| OCCURS integer-1 TO integer-2 TIMES [DEPENDING ON data-name-1]        |
|                                                                       |
|        ( ASCENDING  )                                                 |
|     [  {            } KEY IS data-name-2 [data-name-3] ... ] ...      |
|        ( DESCENDING )                                                 |
|                                                                       |
|           [INDEXED BY index-name-1 [index-name-2] ... ]               |
|                                                                       |
+-----------------------------------------------------------------------+
```

```
+-----------------------------------------------------------------------+
|                            Format 3                                   |
+-----------------------------------------------------------------------+
|                                                                       |
| OCCURS integer-2 TIMES [DEPENDING ON data-name-1]                     |
|                                                                       |
|        ( ASCENDING  )                                                 |
|     [  {            } KEY IS data-name-2 [data-name-3] ... ] ...      |
|        ( DESCENDING )                                                 |
|                                                                       |
|           [INDEXED BY index-name-1 [index-name-2] ... ]               |
|                                                                       |
+-----------------------------------------------------------------------+
```

The other data description clauses associated with an entry whose description includes an OCCURS clause apply to each occurrence of the item described.

Since three subscripts or indexes are allowed, three nested levels of the OCCURS clause are allowed.

The subject of an OCCURS clause is the data-name of the entry that contains this OCCURS clause. The subject of an OCCURS clause must be subscripted or indexed whenever reference is made to it in any statement other than SEARCH.

When subscripted, the subject refers to one occurrence within the table. When not subscripted (permissible only in the SEARCH statement), the subject represents the entire table element. (A table element consists of all occurrences of one level of a table.) A table element must be no greater than 32,767 bytes in length.

The OCCURS clause may not be specified in a data description entry that:

1. has a level-01 or level-77 number

2. describes an item whose size is variable

(The size of an item is variable if the data description of any subordinate item within it contains an OCCURS DEPENDING ON clause--that is, an OCCURS clause with the DEPENDING ON option.)

However, this compiler allows the size of any subordinate item to be variable -- that is, to contain an OCCURS DEPENDING ON clause.

Except for condition-name entries, a record description entry that contains an OCCURS clause must not also contain a VALUE clause.

Within a given record description, the VALUE clause must not be used in a data description entry that is subsequent to a data description entry that contains an OCCURS DEPENDING ON clause.

In the discussion that follows, the term "computational" refers to COMPUTATIONAL, COMPUTATIONAL-1, and COMPUTATIONAL-2 data items.

When a computational elementary item specifies both the OCCURS and SYNCHRONIZED clauses, any necessary slack bytes for each occurrence of the item are added by the compiler. When a group item specifies the OCCURS clause and also contains SYNCHRONIZED computational elementary items, any necessary slack bytes for each occurrence of the group are added by the compiler, as well as the necessary slack bytes for the computational items (see "Slack Bytes" in "Data Division" for a complete discussion).

In Format 1, integer-2 represents the exact number of occurrences. In this case, integer-2 must be greater than zero.

DEPENDING ON OPTION: In Format 2 and Format 3, the DEPENDING ON option is used. This indicates that the subject of this entry has a variable number of occurrences. This does not mean that the length of the subject is variable, but rather that the number of times the subject may be repeated is variable, the number of times being controlled by the value of data-name-1 at object time.

In Format 2, integer-1 represents the minimum number of occurrences, and integer-2 represents the maximum number of occurrences. Integer-1 may be zero or any positive integer. Integer-2 must be greater than zero, and also greater than integer-1. Integer-2 must be less than 32,768. The value of data-name-1 must not exceed integer-2.

In Format 3, integer-2 represents the maximum number of occurrences, and it must be greater than zero and less than 32,768. The value of data-name-1 must not exceed integer-2.

Data-name-1, the object of the DEPENDING ON option:

- must be described as a positive integer

- must not exceed integer-2 in value

- may be qualified, when necessary

- must not be subscripted (that is, must not itself be the subject of, or an entry within, a table)

- must, if it appears in the same record as the table it controls, appear before the variable portion of the record

If the value of data-name-1 is reduced, the contents of data items whose occurrence numbers exceed the new value of data-name-1 become unpredictable.

Unused character positions resulting from the DEPENDING ON option will not appear on external media.

The DEPENDING ON option is required only when the last occurrence of the subject cannot otherwise be determined.

Any Data Division entry that contains an OCCURS DEPENDING ON clause, or which has subordinate to it an entry that contains an OCCURS DEPENDING ON clause, cannot be the object of a REDEFINES clause.

KEY OPTION: The KEY option is used in conjunction with the INDEXED BY option in the execution of a SEARCH ALL statement. The KEY option is used to indicate that the repeated data is arranged in ASCENDING or in DESCENDING order, according to the values contained in data-name-2, data-name-3, etc.

Data-name-2 must be either the name of the entry containing an OCCURS clause, or it must be an entry subordinate to the entry containing the OCCURS clause. If data-name-2 is the subject of this table entry, it is the only key that may be specified for this table. If data-name-2 is not the subject of this table entry, all the keys identified by data-name-2, data-name-3, etc.;

- must be subordinate to the subject of the table entry itself

- must not be subordinate to any other entry that contains an OCCURS clause

- must not themselves contain an OCCURS clause

When the KEY option is specified, the following rules apply:

- Keys must be listed in descending order of significance.

- The total number of keys for a given table element must not exceed 12.

- The sum of the lengths of all keys associated with one table element must not exceed 256 bytes.

- A key may have the following usages: DISPLAY, COMPUTATIONAL-3, or COMPUTATIONAL.

When subordinate entries within the table are variable in length, the following rule also applies:

• Any key in a table element must be at a fixed displacement from the beginning of that element (that is, if a table element is of variable length, then the keys must precede the variable portion).

The following example shows a violation of the last preceding rule:

```
WORKING-STORAGE SECTION.
77  CURRENT-WEEK                          PICTURE 99.
01  TABLE-RECORD.
    02  EMPLOYEE-TABLE OCCURS 100 TIMES
        ASCENDING KEY IS WAGE-RATE EMPLOYEE-NO
        INDEXED BY A, B.
        03  WEEK-RECORD OCCURS 1 TO 52 TIMES
            DEPENDING ON CURRENT-WEEK
            INDEXED BY C.
            04  WEEK-NO                    PIC 99.
            04  AUTHORIZED-ABSENCES        PIC  9.
            04  UNAUTHORIZED-ABSENCES      PIC  9.
            04  LATENESSES                 PIC  9.
        03  EMPLOYEE-NAME                  PIC X(20).
        03  EMPLOYEE-NO                    PIC 9(6).
        03  WAGE-RATE                      PIC 9999V99.
            .
            .
            .
```

WAGE-RATE and EMPLOYEE-NO are invalid as keys, since they are placed after the variable portion of the table.

The following example of the KEY option is correct:

```
WORKING-STORAGE SECTION.
77  CURRENT-WEEK                          PICTURE 99.
01  TABLE-RECORD.
    02  EMPLOYEE-TABLE OCCURS 100 TIMES
        ASCENDING KEY IS WAGE-RATE EMPLOYEE-NO
        INDEXED BY A,   B.
        03  EMPLOYEE-NAME                  PIC X(20).
        03  EMPLOYEE-NO                    PIC 9(6).
        03  WAGE-RATE                      PIC 9999V99.
        03  WEEK-RECORD OCCURS 1 TO 52 TIMES
            DEPENDING ON CURRENT-WEEK
            INDEXED BY C.
            04  WEEK-NO                    PIC 99.
            04  AUTHORIZED-ABSENCES        PIC  9.
            04  UNAUTHORIZED-ABSENCES      PIC  9.
            04  LATENESSES                 PIC  9.
```

The keys WAGE-RATE and EMPLOYEE-NO both appear at a fixed displacement from the beginning of the table element EMPLOYEE-TABLE.

INDEXED BY OPTION: The INDEXED BY option is required if the subject of this entry (the data-name described by the OCCURS clause, or an item within this data-name, if it is a group item) is to be referred to by indexing. The index-name(s) identified by this clause is not defined elsewhere in the program, since its allocation and format are dependent on the system, and, not being data, cannot be associated with any data hierarchy.

The number of index-names for a Data Division entry must not exceed 12.

An <u>index-name</u> must be initialized through a SET statement before it is used.

OCCURS Clause

Each <u>index-name</u> is a fullword in length and contains a binary value that represents an actual displacement from the beginning of the table that corresponds to an occurrence number in the table. The value is calculated as the occurrence number minus one, multiplied by the length of the entry that is indexed by this index-name.

For example, if the programmer writes

A OCCURS 15 TIMES INDEXED BY Z PICTURE IS X(10)

on the fifth occurrence of A, the binary value contained in Z will be:

$$Z = (5 - 1) * 10 = 40$$

Note that, for a table entry of variable length, the value contained in the index-name entry will become invalid when the table entry length is changed, unless the user issues a new SET statement to correct the value contained in the index-name.

The following example of the setting of values in index-name is incorrect:

```
           .
           .
           .
     DATA DIVISION.
           .
           .
           .
     77   E PICTURE S9(5) COMP SYNC.
     01   ...
          02   A OCCURS 10 INDEXED BY IND-1...
               03   B OCCURS 10 DEPENDING ON E INDEXED BY IND-2...
                    .
                    .
                    .
     PROCEDURE DIVISION.
           .
           .
           .
          MOVE 8 TO E
          SET IND-1 TO 3
          SEARCH A ...
               .
               .
               .
          MOVE 10 TO E
          SEARCH A ...
```

(Moving 10 to E changes the length of the table entry A, so that IND-1 now contains an invalid value.)

OCCURS Clause

The following example of the setting of values in index-name is
correct:

```
                      .
                      .
                      .

      DATA DIVISION.
                      .
                      .
                      .

      77  E PICTURE S9(5) COMP SYNC.
      77  D PICTURE S9(5) COMP SYNC.
      01  ...
          02  A OCCURS 10 INDEXED BY IND-1...
              03  B OCCURS 10 DEPENDING ON E INDEXED BY IND-2...
                      .
                      .
                      .

      PROCEDURE DIVISION.
                      .
                      .
                      .

          MOVE 8 TO E
          SET IND-1 TO 3
          SET D TO IND-1
          SEARCH A ...
                      .
                      .
                      .

          MOVE 10 TO E
          SET IND-1 TO D
          SEARCH A ...
                      .
                      .
                      .
```

(Here the user has saved the occurrence number in D, and then later
reset IND-1 to obtain the corrected value.)

There are two types of indexing: Direct Indexing and Relative
Indexing.

Direct Indexing: If a data-name is used in the procedure text with
index-names, the data-name itself must be the subject of an INDEXED BY
option, or be subordinate to a group(s) that is the subject of the
INDEXED BY option.

In the following example

      A (INDEX-1, INDEX-2, INDEX-3)

implies that A belongs to a structure with three levels of OCCURS
options, each with an INDEXED BY option. However, if data-name (A, in
this example) belongs to an OCCURS structure that does not use the
INDEXED BY option, this compiler accepts the specification of
index-names (in this example INDEX-1, INDEX-2, INDEX-3) and assumes the
user has set them to values that correspond to the occurrence number he
wishes to reference.

Relative Indexing: The index-name is followed by a space, followed by one of the operators + or -, followed by another space, followed by an unsigned numeric literal. The numeric literal is considered to be an occurrence number, and is converted to an index value before being added to, or subtracted from, the corresponding index-name.

Given the following example:

A (Z + 1, J + 3, K + 4)

where:

table element indexed by Z has an entry length of 100

table element indexed by J has an entry length of 10

table element indexed by K has an entry length of 2

the resulting address will be computed as follows:

(ADDRESS of A) + Z + 100 * 1 + J + 10 * 3 + K + 4 * 2

conversion of integers
to index values

USAGE IS INDEX Clause

The USAGE IS INDEX clause is used to specify the format of a data item stored internally.

```
r--------------------------------------------------------------------------------¬
|                                   Format                                        |
|--------------------------------------------------------------------------------|
|                                                                                 |
|   [USAGE IS] INDEX                                                              |
|                                                                                 |
L--------------------------------------------------------------------------------
```
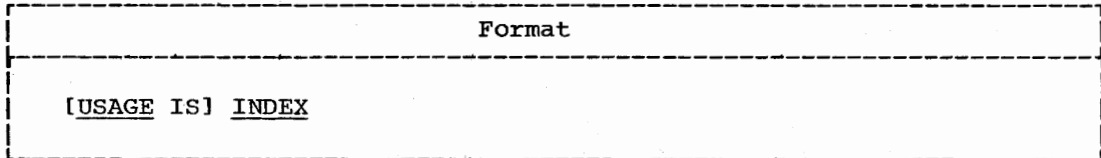
The USAGE IS INDEX clause allows the programmer to specify index data items.

An index data item is an elementary item (not necessarily connected with any table) that can be used to save index-name values for future reference. An index data item must be assigned an index-name value (i.e., (occurrence number - 1) * entry length) through the SET statement. Such a value corresponds to an occurrence number in a table.

The USAGE IS INDEX clause may be written at any level. If a group item is described with the USAGE IS INDEX clause, it is the elementary items within the group that are index data items; the group itself is not an index data item, and the group name cannot be used in SEARCH and SET statements or in relation conditions. The USAGE clause of an elementary item cannot contradict the USAGE clause of a group to which the item belongs.

An index data item can be referred to directly only in a SEARCH or SET statement or in a relation condition. An index data item can be part of a group which is referred to in a MOVE or an input/output

statement. When such operations are executed, however, there is no conversion of the contents of the index data item.

An <u>index data item</u> cannot be a conditional variable.

The SYNCHRONIZED, JUSTIFIED, PICTURE, BLANK WHEN ZERO, or VALUE clauses cannot be used to describe group or elementary items described with the USAGE IS INDEX clause. However, this compiler allows the use of SYNCHRONIZED when USAGE IS INDEX to obtain efficient use of the item.

PROCEDURE DIVISION CONSIDERATIONS FOR TABLE HANDLING

The SEARCH and the SET statements may be used to facilitate table handling. In addition, there are special rules involving Table Handling elements when they are used in relation conditions.

Relation Conditions

Comparisons involving index-names and/or index data items conform to the following rules:

1. The comparison of two index-names is actually the comparison of the corresponding occurrence numbers.

2. In the comparison of an index-name with a data item (other than an index data item), or in the comparison of an index-name with a literal, the occurrence number that corresponds to the value of the index-name is compared with the data item or literal.

3. In the comparison of an index data item with an index-name or another index data item, the actual values are compared without conversion.

Any other comparison involving an index data item is illegal.

Table 25 gives permissible comparisons for index-names and index data items.

Table 25. Index-names and Index Data Items -- Permissible Comparisons

First Operand \ Second Operand	Index-name	Index Data Item	Data-name (numeric integer only)	Numeric literal (integer only)
Index-name	compare occurrence number	compare without conversion	compare occurrence number with data-name	compare occurrence number with literal
Index Data Item	compare without conversion	compare without conversion	illegal	illegal
Data-name (numeric integer only)	compare occurrence number with data-name	illegal	See Table 12 for Permissible Comparisons	
Numeric literal (integer only)	compare occurrence number with literal	illegal		

SEARCH Statement

The SEARCH statement is used to search a table for an element that satisfies a specified condition, and to adjust the value of the associated index-name to the occurrence number corresponding to that table element.

```
+---------------------------------------------------------------+
|                          Format 1                             |
+---------------------------------------------------------------+
|                                                               |
|                            ( index-name-1 )                   |
|    SEARCH identifier-1 [VARYING {              } ]            |
|                            ( identifier-2  )                   |
|                                                               |
|       [AT END imperative-statement-1]                         |
|                                                               |
|                        ( imperative-statement-2 )             |
|       WHEN condition-1 {                        }             |
|                        ( NEXT SENTENCE          )             |
|                                                               |
|                         ( imperative-statement-3 )            |
|       [WHEN condition-2 {                        } ]...       |
|                         ( NEXT SENTENCE          )            |
|                                                               |
+---------------------------------------------------------------+
```

```
┌─────────────────────────────────────────────────────────────────────────┐
│                             Format 2                                      │
├─────────────────────────────────────────────────────────────────────────┤
│                                                                           │
│     SEARCH ALL identifier-1 [AT END imperative-statement-1]               │
│                                                                           │
│                              ⎧ imperative-statement-2 ⎫                    │
│         WHEN condition-1     ⎨                        ⎬                    │
│                              ⎩ NEXT SENTENCE          ⎭                    │
│                                                                           │
└─────────────────────────────────────────────────────────────────────────┘
```

Identifier-1 must not be subscripted or indexed. Its description
must contain an OCCURS clause with the INDEXED BY option. Identifier-1
must not be a floating-point number.

Identifier-1 can be a data item subordinate to a data item that
contains an OCCURS clause, thus providing for a two or three dimensional
table. An index-name must be associated with each dimension of the
table through the INDEXED BY phrase of the OCCURS clause. Execution of
a SEARCH statement causes modification only of the setting of the
index-name associated with identifier-1 (and, if present, of
index-name-1 or identifier-2). Therefore, to search an entire two or
three dimensional table, it is necessary to execute a SEARCH statement
several times; prior to each execution, SET statements must be executed
to adjust the associated index-names to their appropriate settings.

In the AT END and WHEN options, if any of the specified imperative
statement(s) do not terminate with a GO TO statement, control passes to
the next sentence after execution of the imperative statement.

Format 1 Considerations -- Identifier-2, when specified, must be
described as an index data item, or it must be a fixed-point numeric
elementary item described as an integer. When an occurrence number is
incremented, identifier-2 is simultaneously incremented by the same
amount.

Condition-1, condition-2, etc., may be any condition, as follows:

    relation condition
    class condition
    condition-name condition
    sign condition
    switch-status condition
    (condition)
              ⎧ AND ⎫
    [NOT]   ⎨   ⎬ condition
              ⎩ OR ⎭

(See Conditions section of "Procedure Division.")

Upon the execution of a SEARCH statement, a serial search takes
place, starting with the current index setting.

If, at the start of the SEARCH, the value of the index-name
associated with identifier-1 is not greater than the highest possible
occurrence number for identifier-1, the following actions take place:

 1. The condition(s) in the WHEN option are evaluated in the order they
     are written.

2. If none of the conditions are satisfied, the index-name for identifier-1 is incremented to reference the next table element, and step 1 is repeated.

3. If, upon evaluation, one of the WHEN conditions is satisfied, the search terminates immediately, and the imperative-statement associated with that condition is executed. The index-name points to the table element that satisfied the condition.

4. If the end of the table is reached without the WHEN condition being satisfied, the search terminates as described in the next paragraph.

If at the start of the search, the value of the index-name associated with identifier-1 is greater than the highest permissible occurrence number for identifier-1, the search is terminated immediately, and if the AT END option is specified, imperative-statement-1 is executed. If this option is omitted, control passes to the next sentence.

When the VARYING index-name-1 option is specified, one of the following applies:

* If index-name-1 is one of the indexes for identifier-1, index-name-1 is used for the search. Otherwise, the first (or only) index-name for identifier-1 is used.

* If index-name-1 is an index for another table entry, then when the index-name for identifier-1 is incremented to represent the next occurrence of the table, index-name-1 is simultaneously incremented to represent the next occurrence of the table it indexes.

A flowchart of the Format 1 SEARCH operation containing two WHEN options is shown in Chart 5.

Chart 5. Format 1 SEARCH Operation Containing Two WHEN Options

```
                        ****A2*********
                        *   EXECUTION  *
                        *   OF SEARCH  *
                        *    BEGINS    *
                        ***************
                               |
                               V
                          >|
                          V
                        B2 *  *.
*** INDEX SETTING EQUALS  *        *.   GT        AT END *      *****B4**********
    HIGHEST PERMISSIBLE  *.  ***    .*------------------------->*  IMPERATIVE-  *    **
    OCCURRENCE NUMBER     *.       .*                           *  STATEMENT-1  *------->
                            *.   .*                             *               *
                               *. LT OR =                       ****************
                               |
                               V
                        C2 *  *.
                          *        *.   TRUE    WHEN CONDITION-1  *****C4**********
                        *.CONDITION-1.*------------------------->*  IMPERATIVE-  *    **
                          *.       .*                            *  STATEMENT-2  *------->
                            *.   .*                              *               *
                               *.FALSE                           ****************
                               |
                               V
                        D2 *  *.
                          *        *.   TRUE    WHEN CONDITION-2 *  *****D4**********
                        *.CONDITION-2.*------------------------->*  IMPERATIVE-  *    **
                          *.       .*                            *  STATEMENT-3  *------->
                            *.   .*                              *               *
                               *.FALSE                           ****************
                               |
                               V
                        *****E2*********
                        *   INCREMENT   *
                        *INDEX-NAME FOR *
                        * IDENTIFIER-1  *
                        * (INDEX-NAME-1 *
                        * IF APPLICABLE)*
                        ****************
                               |
                               V
                        *****F2*********
                        *   INCREMENT   *
                        * INDEX-NAME-1  *
                        * (FOR ANOTHER  *
                        *  TABLE) OR    *
                        * IDENTIFIER-2  *
                        ****************
```

* THESE OPERATIONS ARE INCLUDED ONLY WHEN CALLED FOR
  IN THE STATEMENT.

** EACH OF THESE CONTROL TRANSFERS IS TO THE NEXT
   SENTENCE UNLESS THE IMPERATIVE-STATEMENT ENDS WITH
   A GO TO STATEMENT.

Format 2 Considerations -- The first index-name assigned to
identifier-1 will be used for the search.

The description of identifier-1 must contain the KEY option in its
OCCURS clause.

Condition-1 must consist of one of the following:

- A relation condition incorporating the EQUALS, EQUAL TO, or equal
  sign (=) relation. Either the subject or the object (but not
  both) of the relation condition must consist solely of one of the
  data-names that appear in the KEY clause of identifier-1.

- A condition-name condition in which the VALUE clause describing the
  condition-name consists of a single literal only. The conditional
  variable associated with the condition-name must be one of the
  data-names that appear in the KEY clause of identifier-1.

- A compound condition formed from simple conditions of the types
  described above, with AND as the only connective.

Any data-name that appears in the KEY clause of identifier-1 may be
tested in condition-1. However, all data-names in the KEY clause
preceding the one to be tested must also be so tested in condition-1.
No other tests may be made in condition-1.

For example, if the following table were defined in the Data
Division:

```
77  VALUE-1 PICTURE 99.
        .
        .
        .
02  A OCCURS 10 TIMES ASCENDING KEY IS KEY1, KEY2, KEY3, KEY4
    INDEXED BY I.
    03   KEY1 PICTURE 9.
    03   KEY2 PICTURE 99.
    03   KEY3 PICTURE 9.
    03   KEY4 PICTURE 9.
         88   BLUE VALUE 1.
        .
        .
        .
```

in the Procedure Division, valid WHEN phrases could be:

```
WHEN KEY1 (I) = 3 AND KEY2 (I) = 10 AND KEY3 (I) = 5 ...

WHEN KEY1 (I) = 3 AND KEY2 (I) = VALUE-1
    AND KEY3 (I) = 5 AND BLUE (I) ...
```

During execution of a Format 2 SEARCH statement, the setting of
index-name is varied during the search so that at no time is it less
than the value that corresponds to the first element of the table, nor
is it ever greater than the value that corresponds to the last element
of the table. If condition-1 cannot be satisfied for any setting of the
index within this permitted range, control is passed to imperative-
statement-1 when the AT END option appears, or to the next sentence when
this clause does not appear. In either case, the final setting of the
index is not predictable. If the index indicates an occurrence that
allows condition-1 to be satisfied, control passes to
imperative-statement-2.

SET Statement

The SET statement establishes reference points for table handling
operations by setting index-names to values associated with table
elements. The SET statement must be used when initializing index-name
values before execution of a SEARCH statement; it may also be used to
transfer values between index-names and other elementary data items.

```
+-----------------------------------------------------------------------+
|                              Format 1                                 |
+-----------------------------------------------------------------------+
|                                                                       |
|      ( index-name-1 [index-name-2]... )        ( index-name-3 )       |
| SET  {                                 }   TO  { identifier-3 }        |
|      ( identifier-1 [identifier-2]...  )        ( literal-1    )       |
|                                                                       |
+-----------------------------------------------------------------------+
```

```
+-----------------------------------------------------------------------+
|                              Format 2                                 |
+-----------------------------------------------------------------------+
|                                                                       |
|                                  ( UP BY   )   ( identifier-4 )        |
| SET index-name-4 [index-name-5] ...{          }{              }       |
|                                  ( DOWN BY )   ( literal-2     )       |
|                                                                       |
+-----------------------------------------------------------------------+
```

All identifiers must name either index data items or fixed-point
numeric elementary items described as integers; however, identifier-4
must not name an index data item. When a literal is used, it must be a
positive integer. Index-names are related to a given table through the
INDEXED BY option of the OCCURS clause; when index-names are specified
in the INDEXED BY option, they are automatically defined.

All references to index-name-1, identifier-1, and index-name-4 apply
equally to index-name-2, identifier-2, and index-name-5, respectively.

Format 1 Considerations -- When the SET statement is executed, one of
the following actions occurs:

1. Index-name-1 is converted to a value that corresponds to the same
    table element to which either index-name-3, identifier-3, or
    literal-1 corresponds. If identifier-3 is an index data item, or
    if index-name-3 is related to the same table as index-name-1, no
    conversion takes place.

2. If identifier-1 is an index data item, it is set equal to either
    the contents of index-name-3 or identifier-3, where identifier-3 is
    also an index data item. Literal-1 cannot be used in this case.

3. If identifier-1 is not an index data item, it is set to an
    occurrence number that corresponds to the value of index-name-3.
    Neither identifier-3 nor literal-1 can be used in this case.

Format 2 Considerations -- When the SET statement is executed, the
contents of index-name-4 (and index-name-5, etc., if present) are
incremented (UP BY) or decremented (DOWN BY) by a value that corresponds
to the number of occurrences represented by the value of literal-2 or
identifier-4.

286   Part V -- Special Features

SAMPLE TABLE HANDLING PROGRAM

The program in Figure 19 illustrates the Table Handling feature, including the use of indexing, of the SET statement, and of the SEARCH statement (including the VARYING option and the SEARCH ALL format).

The census bureau uses the program to compare:

1. the number of births and deaths that occurred in any one of the 50 states in any one of the past 20 years with

2. the total number of births and deaths that occurred in the same state over the entire 20-year period

The input file, INCARDS, contains the specific information upon which the search of the table is to be conducted. INCARDS is formatted as follows:

STATE-NAME   a 4-character alphabetic abbreviation of the state name

SEXCODE   1 = male; 2 = female

YEARCODE   a 4-digit field in the range 1950 through 1969

A typical run might determine the number of females born in New York in 1953 as compared with the total number of females born in New York in the past 20 years.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.   TABLES.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.   IBM-360.
OBJECT-COMPUTER.   IBM-360.
SPECIAL-NAMES.   CONSOLE IS TYPEWRITER.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
     SELECT INFILE ASSIGN TO SYS007-UT-2400-S-INTAPE.
     SELECT OUTFILE ASSIGN TO SYS012-UR-1403-S-PRTOUT.
     SELECT INCARDS ASSIGN TO SYS013-UR-2540R-S-ICARDS.
DATA DIVISION.
FILE SECTION.
FD   INFILE LABEL RECORDS ARE OMITTED.
01   TABLE PIC X(28200).
01   TABLE-2 PIC X(1800).
FD   OUTFILE LABEL RECORDS ARE OMITTED.
01   PRTLINE  PIC X(133).
FD   INCARDS LABEL RECORDS ARE OMITTED.
01   CARDS.
     02   STATE-NAME  PIC X(4).
     02   SEXCODE    PIC 9.
     02   YEARCODE   PIC 9(4).
     02   FILLER     PIC X(71).
WORKING-STORAGE SECTION.
01   PRTAREA-20.
     02   FILLER     PIC X VALUE SPACES.
     02   YEARS-20   PIC 9(4).
     02   FILLER     PIC X(3) VALUE SPACES.
     02   BIRTHS-20 PIC 9(7).
     02   FILLER     PIC X(3) VALUE SPACES.
     02   DEATHS-20 PIC 9(7).
     02   FILLER     PIC X(108) VALUE SPACES.
```

Figure 19.   Sample Table Handling Program (Part 1 of 2)

```
01   PRTAREA.
     02   FILLER     PIC X.
     02   YEAR       PIC 9(4).
     02   FILLER     PIC X(3) VALUE SPACES.
     02   BIRTHS     PIC 9(5).
     02   FILLER     PIC X(3) VALUE SPACES.
     02   DEATHS     PIC 9(5).
     02   FILLER     PIC X(112) VALUE SPACES.
01   CENSUS-STATISTICS-TABLE.
     02   STATE-TABLE OCCURS 50 TIMES INDEXED BY ST.
          03   STATE-ABBREV    PIC X(4).
          03   SEX OCCURS 2 TIMES INDEXED BY SE.
               04   STATISTICS OCCURS 20 TIMES ASCENDING KEY IS YEAR
                    INDEXED BY YR.
                    05   YEAR     PIC 9(4).
                    05   BIRTHS   PIC 9(5).
                    05   DEATHS   PIC 9(5).
01   STATISTICS-LAST-20-YRS.
     02   SEX-20 OCCURS 2 TIMES  INDEXED BY SE-20.
          03   STATE-20 OCCURS 50 TIMES INDEXED BY ST-20.
               04   YEARS-20     PIC 9(4).
               04   BIRTHS-20    PIC 9(7).
               04   DEATHS-20    PIC 9(7).
PROCEDURE DIVISION.
OPEN-FILES.
     OPEN INPUT INFILE INCARDS OUTPUT OUTFILE.
READ-TABLE.
     READ INFILE INTO CENSUS-STATISTICS-TABLE
          AT END GO TO READ-CARDS.
     READ INFILE INTO STATISTICS-LAST-20-YRS
          AT END GO TO READ-CARDS.
READ-CARDS.
     READ INCARDS
          AT END GO TO EOJ.
DETERMINE-ST.
     SET ST ST-20 TO 1.
     SEARCH STATE-TABLE VARYING ST-20 AT END GO TO ERROR-MSG-1
          WHEN STATE-NAME = STATE-ABBREV (ST) NEXT SENTENCE.
DETERMINE-SE.
     SET SE SE-20 TO SEXCODE.
DETERMINE-YR.
     SEARCH ALL STATISTICS AT END GO TO ERROR-MSG-2
          WHEN YEAR OF STATISTICS (ST, SE, YR) = YEARCODE
               GO TO WRITE-RECORD.
ERROR-MSG-1.
     DISPLAY "INCORRECT STATE "  STATE-NAME UPON TYPEWRITER.
     GO TO READ-CARDS.
ERROR-MSG-2.
     DISPLAY "INCORRECT YEAR "   YEARCODE UPON TYPEWRITER.
     GO TO READ-CARDS.
WRITE-RECORD.
     MOVE CORRESPONDING STATISTICS (ST, SE, YR) TO PRTAREA.
     WRITE PRTLINE FROM PRTAREA AFTER ADVANCING 3.
     MOVE CORRESPONDING STATE-20 (SE-20, ST-20) TO PRTAREA-20.
     WRITE PRTLINE FROM PRTAREA-20 AFTER ADVANCING 1.
     GO TO READ-CARDS.
EOJ.
     CLOSE INFILE INCARDS OUTFILE.
     STOP RUN.
```

Figure 19. Sample Program for the Table Handling Feature (Part 2 of 2)

The segmentation feature allows the problem programmer to specify object program overlay requirements. The segmentation feature permits segmentation of procedures only. The Procedure Division and Environment Division are considered in determining segmentation requirements for an object program.

ORGANIZATION

Although it is not mandatory, the Procedure Division for a source program is usually written as several consecutive sections, each of which is composed of a series of closely related operations that are designed to perform collectively a particular function. However, when segmentation is used, the entire Procedure Division must be in sections. In addition, each section must be classified as belonging either to the fixed portion or to one of the independent segments of the object program. Segmentation in no way affects the need for qualification of procedure-names to ensure uniqueness.

FIXED PORTION

The fixed portion is defined as that part of the object program that is logically treated as if it were always in computer storage. This portion of the program is composed of two types of computer storage segments, permanent segments and overlayable fixed segments.

A permanent segment is a segment in the fixed portion that cannot be overlaid by any other part of the program.

An overlayable fixed segment is a segment in the fixed portion which, although logically treated as if it were always in storage, can be overlaid (if necessary) by another segment to optimize storage utilization. However, such a segment, if called for by the program, is always made available in the state it was in when it was last used.

Depending on the availability of storage, the number of permanent segments in the fixed portion can be varied through the use of a special facility called SEGMENT-LIMIT, which is discussed in "Structure of Program Segments."

INDEPENDENT SEGMENTS

An independent segment is defined as that part of the object program which can overlay, and be overlaid by, either an overlayable fixed segment or another independent segment. An independent segment is always considered to be in its initial state each time it is made available to the program.

Segmentation Control and Structure

SEGMENT CLASSIFICATION

Sections that are to be segmented are classified by means of a system of priority numbers. The following criteria should be used:

- **Logical requirements**: Sections that must be available for reference at all times, or which are referred to very frequently, are normally classified as belonging to one of the permanent segments; sections that are less frequently used are normally classified as belonging either to one of the overlayable fixed segments or to one of the independent segments, depending on logic requirements.

- **Frequency of use**: Generally, the more frequently a section is referred to, the lower its priority number should be; the less frequently it is referred to, the higher its priority number should be.

- **Relationship to other sections**: Sections that frequently communicate with one another should be given equal priority numbers. All sections with the same priority number constitute a single program segment.

SEGMENTATION CONTROL

The logical sequence of the program is the same as the physical sequence except for specific transfers of control. A reordering of the object module will be necessary if a given segment has its sections scattered throughout the source program. However, the compiler will provide transfers to maintain the logic flow of the source program. The compiler will also insert instructions necessary to load and/or initialize a segment when necessary. Control may be transferred within a source program to any paragraph in a section; that is, it is not mandatory to transfer control to the beginning of a section.

STRUCTURE OF PROGRAM SEGMENTS

PRIORITY NUMBERS

Section classification is accomplished by means of a system of priority numbers. The priority number is included in the section header.

```
+-----------------------------------------------------------------------+
|                              Format                                   |
+-----------------------------------------------------------------------+
|                                                                       |
|    section-name SECTION [priority-number].                            |
|                                                                       |
+-----------------------------------------------------------------------+
```

All sections that have the same priority-number constitute a program segment with that priority.

The priority-number must be an integer ranging in value from 0 through 99.

Segments with priority-numbers 0 through 49 belong to the fixed portion of the object program.

Segments with priority-numbers 50 through 99 are independent segments.

Sections in the declaratives portion of the Procedure Division must not contain priority-numbers in their section headers. They are treated as fixed segments with a priority-number of zero.

If the priority-number is omitted from the section header, the priority is assumed to be zero.

When a procedure-name in an independent segment 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 for each execution of the PERFORM statement.

SEGMENT LIMIT

Ideally, all program segments having priority-numbers ranging from 0 through 49 are treated as permanent segments. However, when insufficient storage is available to contain all permanent segments plus the largest overlayable segment, it becomes necessary to decrease the number of permanent segments. The SEGMENT-LIMIT feature provides the user with a means by which he can reduce the number of permanent segments in his program, while these permanent segments still retain the logical properties of fixed portion segments (priority numbers 0 through 49).

```
r-------------------------------------------------------------------1
|                             Format                                |
|-------------------------------------------------------------------|
|                                                                   |
|    [SEGMENT-LIMIT IS priority-number]                             |
|                                                                   |
L-------------------------------------------------------------------J
```

The SEGMENT-LIMIT clause is coded in the OBJECT-COMPUTER paragraph.

Priority-number must be an integer that ranges in value from 1 through 49.

When the SEGMENT-LIMIT clause is specified, only those segments having priority-numbers from 0 up to, but not including, the priority number designated as the segment limit are considered as permanent segments of the object program.

Those segments having priority numbers from the segment limit through 49 are considered as overlayable fixed segments.

When the SEGMENT-LIMIT clause is omitted, all segments having priority numbers from 0 through 49 are considered to be permanent segments of the object program.

RESTRICTIONS ON PROGRAM FLOW

When segmentation is used, the following restrictions are placed on the ALTER and PERFORM statements and on called programs:

ALTER Statement

1. A GO TO statement in a section whose priority number is 50 or higher must not be referred to by an ALTER statement in a section with a different priority number.

2. A GO TO statement in a section whose priority number is lower than 50 may be referred to by an ALTER statement in any section, even if the GO TO statement to which the ALTER refers is in a segment of the program that has not yet been called for execution.

PERFORM Statement

1. A PERFORM statement that appears in a section whose priority number is lower than the segment limit can have within its range only the following:

    a. Sections with priority numbers lower than 50.

    b. Sections wholly contained in a single segment whose priority number is higher than 49.

    However, this compiler allows the PERFORM to have within its range sections with any priority numbers.

2. A PERFORM statement that appears in a section whose priority number is equal to or higher than the segment limit can have within its range only the following:

    a. Sections with the same priority number as the section containing the PERFORM statement.

    b. Sections with priority numbers that are lower than the segment limit.

    However, this compiler allows the PERFORM to have within its range sections with any priority numbers.

When a procedure-name in a permanent segment is referred to by a PERFORM statement in an independent segment, the independent segment is reinitialized upon exit from the performed paragraphs.

Called Programs

A called program may not be segmented.

SOURCE PROGRAM LIBRARY FACILITY

Prewritten source program entries can be included in a source program at compile time. Thus, an installation can use standard file descriptions, record descriptions, or procedures, without recoding them. These entries and procedures are contained in user-created libraries; they are included in a source program by means of a COPY statement.

COPY Statement

The COPY statement permits the user to include prewritten Data Division entries, Environment Division clauses, and Procedure Division procedures in his source program.

```
---------------------------------------------------------------------------
|                            General Format                               |
---------------------------------------------------------------------------
| Option 1 (within the Configuration Section):                            |
|                                                                         |
|    SOURCE-COMPUTER.   COPY statement.                                    |
|    OBJECT-COMPUTER.   COPY statement.                                    |
|    SPECIAL-NAMES.   COPY statement.                                      |
---------------------------------------------------------------------------
| Option 2 (within the Input-Output Section):                             |
|                                                                         |
|    FILE-CONTROL.   COPY statement.                                      |
|    I-O-CONTROL.   COPY statement.                                       |
---------------------------------------------------------------------------
| Option 3 (within the File-Control Paragraph):                           |
|                                                                         |
|    SELECT file-name COPY statement.                                     |
---------------------------------------------------------------------------
| Option 4 (within the File Section):                                     |
|                                                                         |
|    FD  file-name COPY statement.                                        |
|    SD  sort-file-name COPY statement.                                   |
---------------------------------------------------------------------------
| Option 5 (within the Report Section):                                   |
|                                                                         |
|    RD  report-name COPY statement.                                      |
|    RD  report-name [WITH CODE mnemonic-name] COPY statement.            |
---------------------------------------------------------------------------
| Option 6 (within a File or Sort description entry, or within the        |
|          Working-Storage Section or the Linkage Section):               |
|                                                                         |
|    01  data-name COPY statement.                                        |
---------------------------------------------------------------------------
| Option 7 (with a Report Group):                                         |
|                                                                         |
|    01  [data-name] COPY statement.                                      |
---------------------------------------------------------------------------
| Option 8 (within the Working-Storage Section or the Linkage Section):   |
|                                                                         |
|    77  data-name COPY statement.                                        |
---------------------------------------------------------------------------
| Option 9 (within the Procedure Division):                               |
|                                                                         |
|    section-name SECTION [priority-number].   COPY statement.            |
|    paragraph-name.   COPY statement.                                    |
---------------------------------------------------------------------------
```

COPY Statement

```
r-----------------------------------------------------------------------------1
|                                   Format                                     |
|-----------------------------------------------------------------------------|
|                                                                             |
|   COPY library-name [SUPPRESS]                                              |
|                                                                             |
|                                        ( word-2      )                      |
|          [REPLACING   word-1   BY  { literal-1    }                         |
|                                        ( identifier-1)                      |
|                                                                             |
|                              ( word-4      )                                |
|             [word-3   BY  { literal-2    }      ]...].                      |
|                              ( identifier-2 )                               |
|                                                                             |
L-----------------------------------------------------------------------------J
```

No other statement or clause may appear in the same entry as the COPY statement, with the exception of the Report Description entry and the File-Control Paragraph.

When the library text is copied from the library, compilation is the same as though the text were actually part of the source program.

The COPY statement processing is terminated by the end of the library text.

The text contained in the library must not contain any COPY statements.

Library-name is the name of the library text contained in the user's library. Library-name must follow the rules of formation for program-name. The first eight characters are used as the identifying name.

The words preceding COPY conform to margin restrictions for COBOL programs. On a given source program card containing the completion of a COPY statement, there must be no information beyond the statement terminating period. The material introduced into the source program by the COPY statement will follow the COPY statement on the listing beginning on the next line. However, the SUPPRESS option may be used to indicate that the library entry is not to be listed.

If the REPLACING option is used, each word specified in the format is replaced by the stipulated word, identifier, or literal which is associated with it in the format.

Word-1, word-2, etc., may be a data-name, procedure-name, condition-name, mnemonic-name, or file-name.

Use of the REPLACING option does not alter the material as it appears in the library.

When options 1, 2, 3, 4, 5, or 9 are written, the words COPY library-name are replaced by the information identified by library-name. This information comprises the sentences or clauses needed to complete the paragraph, sentence, or entry containing the COPY statement.

When options 6, 7, or 8 are written, the entire entry is replaced by the information identified by library-name, except that data-name (if specified) replaces the corresponding data-name from the library.

For example, if the library entry PAYLIB consists of the following Data Division record:

```
01  A.
    02  B               PIC S99.
    02  C               PIC S9(5)V99.
    02  D               PIC S9999 OCCURS 0 TO 52 TIMES
            DEPENDING ON B OF A.
```

the programmer can use the COPY statement in the Data Division of his program as follows:

```
01  PAYROLL COPY PAYLIB.
```

In this program, the library entry is then copied as follows:

```
01  PAYROLL.
    02  B               PIC S99.
    02  C               PIC S9(5)V99.
    02  D               PIC S9999 OCCURS 0 TO 52 TIMES
            DEPENDING ON B OF A.
```

Note that the data-name A has not been changed in the DEPENDING ON option.

To change some (or all) of the names within the library entry to names he wishes to reference within his program, the programmer can use the REPLACING option:

```
01  PAYROLL COPY PAYLIB REPLACING A BY PAYROLL
        B BY PAY-CODE C BY GROSSPAY.
```

In this program, the library entry is copied as follows:

```
01  PAYROLL.
    02  PAY-CODE        PIC S99.
    02  GROSS-PAY       PIC S(5)V99.
    02  D               PIC S9999 OCCURS 0 TO 52 TIMES
            DEPENDING ON PAY-CODE OF PAYROLL.
```

The changes shown are made only for this program. The entry as it appears in the library remains unchanged.

EXTENDED SOURCE PROGRAM LIBRARY FACILITY

A complete program may be included as an entry in the user's library and may be used as the basis of compilation. Input to the compiler is a BASIS card, followed by any number of INSERT and/or DELETE cards, followed by any number of debugging packets, if desired (see "Debugging Language").

BASIS Card

```
Format
----------------------------------------------------------------

BASIS   library-name
```

The word BASIS followed by library-name may appear anywhere within columns 1 through 72 on the card. There must be no other text on the card.

Library-name must follow the rules of formation for program-name. It is the name by which the library entry is known to the control program. The first eight characters are used as the identifying name.

If the INSERT or DELETE cards follow the BASIS card, the library entry is modified prior to being processed by the compiler. Use of INSERT or DELETE cards does not alter the material in the library.

INSERT Card

```
Format
----------------------------------------------------------------

INSERT sequence-number-field
```

DELETE Card

```
Format
----------------------------------------------------------------

DELETE sequence-number-field
```

The word INSERT or DELETE, followed by a space, followed by sequence-number-field may appear anywhere within columns 1 through 72 on the card. There must be no other text on the card.

Each number in sequence-number-field must refer to a sequence number of the basic library entry. The sequence-number is the six-digit number the programmer assigns in columns 1 through 6 of the COBOL Coding Form.

The numbers in the sequence-number-field must be in ascending numerical order from the first INSERT/DELETE card to the last INSERT/DELETE card in the program.

The sequence-number-field of an INSERT card must be a single number (e.g., 200310). At least one new source program card must follow the INSERT card for insertion after the card specified by the sequence-number-field.

The entries comprising sequence-number-field of a DELETE card must be numbers or ranges of numbers. Each entry must be separated from the preceding entry by a comma followed by a space. Ranges of numbers are indicated by separating the two bounding numbers of the range by a hyphen. For example:

    000001-000005, 000010

Source program cards may follow a DELETE card, for insertion before the card following the last one deleted.

The following statements are provided for program debugging. They
may appear anywhere in an IBM American National Standard COBOL program
or in a compile-time debugging packet.

For the TRACE and EXHIBIT statements, the output is written on the
system list device (SYSLST).

READY/RESET TRACE Statement

```
r-----------------------------------------------------------------------------------
|                                     Format                                        |
|-----------------------------------------------------------------------------------|
|                                                                                   |
|   ( READY )                                                                       |
|   {       }   TRACE                                                               |
|   ( RESET )                                                                       |
|                                                                                   |
L-----------------------------------------------------------------------------------
```

After a READY TRACE statement is executed, each time execution of a
paragraph or section begins, its compiler-generated card number is
displayed.

The execution of a RESET TRACE statement terminates the functions of
a previous READY TRACE statement.

EXHIBIT Statement

```
r-----------------------------------------------------------------------------------
|                                     Format                                        |
|-----------------------------------------------------------------------------------|
|                                                                                   |
|              ( NAMED                  ) ( identifier-1            )                |
|   EXHIBIT    { CHANGED NAMED          ) {                          }               |
|              ( CHANGED                ) ( nonnumeric-literal-1     )               |
|                                                                                   |
|         [ identifier-2           ]                                                |
|         [                        ] ...                                            |
|         [ nonnumeric-literal-2   ]                                                |
|                                                                                   |
L-----------------------------------------------------------------------------------
```

The execution of an EXHIBIT statement causes a formatted display of
the identifiers (or nonnumeric literals) listed in the statement.

Identifiers listed in the statement cannot be any special register
except TALLY.

Nonnumeric-literals listed in the statement are followed by a space
when displayed.

The display of the operands is continued as described for the DISPLAY statement. A maximum logical record size of 120 characters is assumed.

EXHIBIT NAMED: Each time an EXHIBIT NAMED statement is executed, there is a formatted display of each identifier listed and its value. Since both the identifying name and the value of the identifier are displayed, a fixed columnar format is unnecessary. If the list of operands includes nonnumeric-literals, they are displayed as remarks each time the statement is executed.

The format of the output for each identifier listed in the EXHIBIT NAMED statement is:

> original identifying name, including qualifiers if written (no more than 120 characters in length)
> space
> equal sign
> space
> value of identifier (no more than 256 bytes in length)
> space

EXHIBIT CHANGED NAMED: Each time an EXHIBIT CHANGED NAMED statement is executed, there is a display of each identifier listed and its value only if the value has changed since the previous time the statement was executed. The initial time such a statement is executed, all values are considered changed and are displayed. If the list of operands includes nonnumeric-literals, they are displayed as remarks each time the statement is executed.

Since both the identifying name and the value of each identifier is displayed, a fixed columnar format is unnecessary. If some of the identifiers have not changed in value, no space is reserved for them. If none of the identifiers have changed in value, no blank line(s) will be printed.

The format of the output for each identifier listed in the EXHIBIT CHANGED NAMED statement is:

> original identifying name, including qualifiers if written (no more than 120 characters in length)
> space
> equal sign
> space
> value of identifier (no more than 256 bytes in length)
> space

EXHIBIT CHANGED: Each time an EXHIBIT CHANGED statement is executed, there is a display of the current value of each identifier listed only if the value has changed since the previous time the statement was executed. The initial time the statement is executed, all values are considered changed and are displayed. If the list of operands includes nonnumeric-literals, they are printed as remarks each time the statement is executed.

The format of the output for a specific EXHIBIT CHANGED statement presents each operand in a fixed columnar position. Since the operands are displayed in the order they are listed in the statement, the programmer can easily distinguish each operand. The following considerations apply:

• If there are two or more identifiers as operands, and some, but not all, are changed from the previous execution of the statement, only the changed values are displayed. The positions reserved for a given operand are blank when the value of the operand has not changed.

- If none of the operands have changed in value from the previous
  execution of the statement, a blank line(s) will be printed.

- Variable length identifiers are not permitted as operands.

- The storage reserved for any operand cannot exceed 256 bytes.

<u>Note</u>: The combined total length of all operands for all EXHIBIT CHANGED
NAMED plus all EXHIBIT CHANGED statements in one program cannot exceed
32,767 bytes.

If two distinct EXHIBIT CHANGED NAMED or two EXHIBIT CHANGED
statements appear in one program, each specifying the same <u>identifiers</u>,
the changes in value of those identifiers are associated with each of
the two separate statements. Depending on the path of program flow, the
values of the identifier saved for comparison may differ for each of the
two statements.

ON (Count-conditional) Statement

The ON statement allows the programmer to specify when the statements
it contains are to be executed.

```
---------------------------------------------------------------------
|                              Format 1                             |
|-------------------------------------------------------------------|
|                                                                   |
|  ON integer-1 [AND EVERY integer-2] [UNTIL integer-3]             |
|                                                                   |
|   { imperative-statement ...}   { ELSE     }   { statement ...  } |
|   { NEXT SENTENCE           }   { OTHERWISE }   { NEXT SENTENCE  } |
|                                                                   |
---------------------------------------------------------------------
```

```
---------------------------------------------------------------------
|                       Format 2 (Version 3)                        |
|-------------------------------------------------------------------|
|                                                                   |
|        { integer-1    }            { integer-2    }               |
|   ON   {              } [AND EVERY {              } ]             |
|        { identifier-1 }            { identifier-2 }               |
|                                                                   |
|             { integer-3    }    { imperative-statement }          |
|   [UNTIL   {              } ]   {                       }          |
|             { identifier-3 }    { NEXT SENTENCE         }          |
|                                                                   |
|   { ELSE      }   { statement ...  }                              |
|   { OTHERWISE }   { NEXT SENTENCE  }                              |
|                                                                   |
---------------------------------------------------------------------
```

All <u>integers</u> specified in the ON statement must be positive and no
greater than 16,777,215.

The phrase ELSE/OTHERWISE NEXT SENTENCE may be omitted if it
immediately precedes the period for the sentence.

Program Product Information -- Version 3

All identifiers must be fixed-point numeric items described as integers. Their values must be positive and no greater than 16,777,215.

At object time each identifier must be initialized to a positive value before the first execution of the ON statement. Between executions of the ON statement, the values contained in the identifiers may be modified. The programmer's manipulation of these values in no way affects the compiler-generated counters associated with the ON statement.

In the following discussion, each reference to integer-1 applies equally to identifier-1. Similarly, each reference to integer-2 applies to identifier-2, and each reference to integer-3 applies to identifier-3.

In all versions of the compiler, the ON statement is evaluated and executed as follows:

- Each ON statement has compiler-generated counters and save areas associated with it. The counters are initialized in the object program. Each time the path of program flow reaches the ON statement, the counters are incremented, and the count-condition (integer-1 AND EVERY integer-2 UNTIL integer-3) is tested.

- If the count-condition is satisfied, the imperative-statement (or NEXT SENTENCE) preceding ELSE/OTHERWISE is executed. (Note that an imperative-statement may consist of a series of imperative statements.)

- If the count-condition is not satisfied, the statement(s) (or NEXT SENTENCE) following ELSE/OTHERWISE is executed. If the ELSE/OTHERWISE option does not appear, the next sentence is executed.

The count-condition is evaluated as follows:

- If only integer-1 has been specified, then the count-condition is satisfied only once: when the path of program flow has reached the ON statement integer-1 times -- that is, when the value in the counter equals integer-1.

- When only integer-1 and integer-3 are specified, then the value of integer-2 is assumed to be one, and the count-condition is satisfied when the value in the counter is any value within the range integer-1 through integer-3.

- If only integer-1 and integer-2 are specified, then the count-condition is satisfied each time the value in the counter is equal to integer-1 + (integer-2 * K), where K is any positive integer or zero. No upper limit for the execution of the ON statement is assumed.

- When all three integers are specified, then the count-condition is satisfied as in the last preceding case, except that an upper limit beyond which the count-condition cannot be satisfied is specified. The upper limit is integer-3.
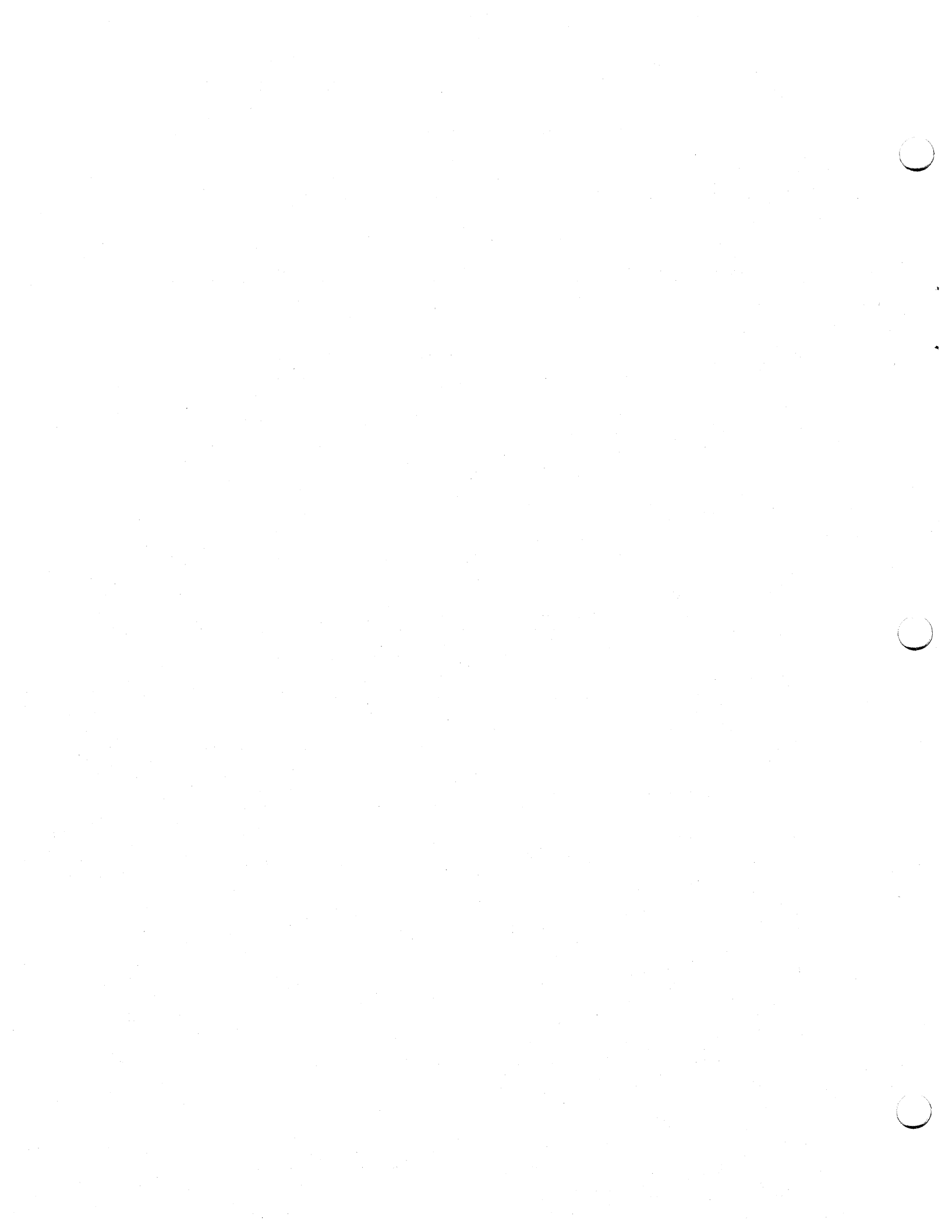
COMPILE-TIME DEBUGGING PACKET

Debugging statements for a given paragraph or section in a program may be grouped together into a debugging packet. These statements will be compiled with the source language program and will be executed at object time. Each packet refers to a specified paragraph-name or section-name in the Procedure Division. Compile-time debugging packets are grouped together and are placed immediately following the source program. No reference to procedure-names in debug packets may be made in the body of the program.

DEBUG Card

Each compile time debug packet is headed by the control card DEBUG.

```
---------------------------------------------------------------------------
|                               Format                                      |
|---------------------------------------------------------------------------|
|                                                                           |
|   DEBUG   location                                                        |
|                                                                           |
---------------------------------------------------------------------------
```

The word DEBUG followed by location may appear anywhere within columns 1 through 72 on the card. There must be no other text on the card.

The location is the section-name or paragraph-name (qualified, if necessary) indicating the point in the program at which the packet is to be executed. Effectively, the statements in the packet are executed as though they were physically placed in the source program following the section-name or paragraph-name, but preceding the text associated with the procedure. The same location must not be used in more than one DEBUG control card. Location cannot be a paragraph-name within any DEBUG packet.

A debug packet may consist of any procedural statements conforming to the requirements of COBOL. The following considerations apply:

• A PERFORM or ALTER statement in a debug packet may refer to a procedure-name in any debug packet or in the main body of the Procedure Division.

• A GO TO statement in a debug packet may not refer to a procedure-name in another debug packet, but it may refer to a procedure-name in the main body of the Procedure Division.

FORMAT CONTROL OF THE SOURCE PROGRAM LISTING

There are four statements that allow the programmer to control the spacing of the source program listings produced by the COBOL compiler. These statements are: EJECT, SKIP1, SKIP2, and SKIP3. They may be written anywhere in the source program.

EJECT Statement

The EJECT statement instructs the compiler to print the next source statement at the top of the next page.

```
---------------------------------------------------------------------
|                              Format                               |
|-------------------------------------------------------------------|
|                                                                   |
|    1          Area B                                              |
|          ---------------------                                    |
|               EJECT                                               |
|                                                                   |
---------------------------------------------------------------------
```

The word EJECT may be written anywhere within Area B and must be the only statement on the card. There must be no punctuation.

SKIP1, SKIP2, and SKIP3 Statements

These statements instruct the compiler to skip 1, 2, or 3 lines before printing the next source statement.

```
---------------------------------------------------------------------
|                              Format                               |
|-------------------------------------------------------------------|
|                                                                   |
|    1          Area B                                              |
|          -------- ----------                                      |
|               ( SKIP1 )                                           |
|               { SKIP2 }                                           |
|               ( SKIP3 )                                           |
|                                                                   |
---------------------------------------------------------------------
```

SKIP1 tells the compiler to skip 1 line (double spacing).

SKIP2 tells the compiler to skip 2 lines (triple spacing).

SKIP3 tells the compiler to skip 3 lines (quadruple spacing).

SKIP1, SKIP2, or SKIP3 may be written anywhere within Area B and must be the only statement on the card. There must be no punctuation.

COBOL provides facilities for handling sterling currency items by means of an extension of the PICTURE clause. Additional options and formats, necessitated by the nondecimal nature of sterling and by the conventions by which sterling amounts are represented in punched cards, are also available.

COBOL provides a means to express sterling currency in pounds, shillings, and pence, in that order. There are 20 shillings in a pound, and 12 pence in a shilling. Although sterling amounts are sometimes expressed in shillings and pence only (in which case the number of shillings may exceed 99), within machine systems, shillings will always be expressed as a two-digit field. Pence, when in the form of integers, likewise will be expressed as a two-digit field. However, provision must be made for pence to be expressed as decimal fractions as well, as in the form 17s.10.237d.

The IBM method for representing sterling amounts in punched cards uses two columns for shillings and one for pence. Tenpence (10d.) is represented by an '11' punch and elevenpence (11d.) by a '12' punch. The British Standards Institution (B.S.I.) representation uses single columns for both shillings and pence. The B.S.I. representation for shillings consists of a '12' punch for ten shillings and the alphabetic punches A through I for 11 through 19 shillings, respectively.

Note: The B.S.I. representation for shillings precludes the use of more than 19 shillings in a sterling expression; therefore, 22/10 (that is, 22 shillings 10 pence) must be expanded by the user to 1/2/10. Similarly, the guinea -- 21 shillings -- or any multiple thereof, must be expanded to pounds and shillings.

The indicated representations may be used separately or in combination, resulting in four possible conventions.

1. IBM shillings and IBM pence

2. IBM shillings and B.S.I. pence

3. B.S.I. shillings and IBM pence

4. B.S.I. shillings and B.S.I. pence

Any of these conventions may be associated with any number of digits (or none) in the pound field and any number of decimal places (or none) in the pence field. In addition, sign representation may be present as an overpunch in one of several allowable positions in the amount, or may be separately entered from another field.

Two formats are provided in the PICTURE clause for the representation of sterling amounts: sterling report format (used for editing) and sterling nonreport format (used for arithmetic).

Sterling Nonreport Format

In the formats that follow, n stands for a positive integer other than zero. This integer enclosed in parentheses and following the symbols 9, B, etc., indicates the number of consecutive occurrences of the preceding symbol. For example, 9(6) and 999999 are equivalent. The PICTURE characters used to describe sterling items are:

    6 7 8 9 C D * , / B Z V . £ : s d CR DB + -

(The character £ is the sterling equivalent of the character $.)

<u>Note</u>: The lower-case letters "s" and "d" are represented by an 11-0-2 punch and a 12-0-4 punch, respectively.

STERLING NONREPORT

The format of the PICTURE clause for a sterling nonreport data item is:

```
-------------------------------------------------------------------
|                               Format                            |
|-----------------------------------------------------------------|
|  ( PICTURE )                    ( 6[6] )                         |
|  {         } IS 9[(n)]D[8]8D    {      } [[V]9[(n)]] [USAGE IS] DISPLAY-ST |
|  ( PIC     )                    ( 7[7] )                         |
-------------------------------------------------------------------
```
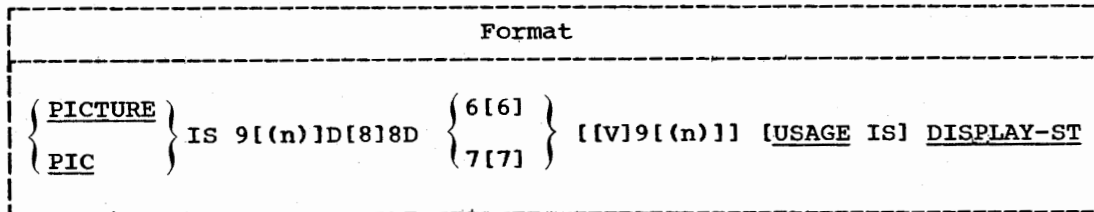
<u>Note</u>: For a sterling nonreport picture to be valid, it must contain a pound field, a shilling field, and a pence field.

The representation for pounds is 9[(n)]D where:

1. The character 9 indicates that a character position will always contain a numeric character, and may extend to n positions.

2. The character D indicates the position of an assumed pound separator.

The representation for shillings is [8]8D where:

1. The characters [8]8 indicate the position of the shilling field and the convention by which shillings are represented in punched cards. 88 indicates IBM shilling representation occupying a two-column field. 8 indicates B.S.I. single-column shilling representation.

2. The character D indicates the position of an assumed shilling separator.

The representation for pence is:

  (6[6])
  {   } [[V]9[(n)]]
  (7[7])

1. The character 6 indicates IBM single-column pence representation wherein 10d. is represented by an '11' punch and 11d. by a '12' punch. The characters 66 indicate two-column representation of pence, usually from some external medium other than punched cards.

2. The character 7 indicates B.S.I. single-column pence representation wherein 10d. is represented by a '12' punch and 11d. by an '11' punch. The characters 77 indicate two-column representation of pence. Consequently, 66 and 77 serve the same purpose and are interchangeable.

3. The character V indicates the position of an assumed decimal point in the pence field. Its properties and use are identical with that of V in dollar amounts. Decimal positions in the pence field may extend to n positions.

4. The character 9 indicates that a character position will always contain a numeric character, and may extend to n positions.

Example: Assume that a sterling currency data item used in arithmetic expressions is to be represented in IBM shillings and IBM pence, and that this data item will never exceed 99/19s/11d. Its picture should be:

    PICTURE 9(2)D88D6 DISPLAY-ST.

The VALUE clause must not be specified for sterling nonreport items.

Sterling Sign Representation

Signs for sterling amounts may be entered as overpunches in one of several allowable positions of the amount. A sign is indicated by an embedded S in the nonreport PICTURE immediately to the left of the position containing the overpunch. Allowable overpunch positions are the high-order and low-order positions of the pound field, the high-order shilling digit in two-column shilling representation, the low-order pence digit in two-column pence representation, or the least significant decimal position of pence.

The following are examples of sterling currency nonreport data items showing sign representation in each of the allowable positions:

    PICTURE S99D88D6V9(3) DISPLAY-ST

    PICTURE 9S9D88D6V9(3) DISPLAY-ST

    PICTURE 9(2)DS88D6V9(3) DISPLAY-ST

    PICTURE 9(2)D88D6S6V9(3) DISPLAY-ST

    PICTURE 9(2)D88D6V99S9 DISPLAY-ST

STERLING REPORT

The sterling currency report data is composed of four portions: pounds, shillings, pence, and pence decimal fractions.

```
r----------------------------------------------------------------------------1
|                                    FORMAT                                   |
|----------------------------------------------------------------------------|
| ( PICTURE )                                                                 |
| {         }  IS                                                             |
| ( PIC     )                                                                 |
|                                                                             |
|   [pound-report-string][pound-separator-string]delimiter                    |
|                                                                             |
|    shilling-report-string[shilling-separator-string]delimiter               |
|                                                                             |
|    pence-report-string[pence-separator-string][sign-string]                 |
|                                                                             |
|   [USAGE IS] DISPLAY-ST                                                      |
L----------------------------------------------------------------------------J
```

Pound-Report-String - This string is optional. It is subject to the same rules as other numeric edited items, with the following exceptions:

* The allowable characters are: £ (pound symbol) 9 Z * + - 0 (zero) B , (comma).

* The total number of digits in the pound-report-string plus the fractional-pence field cannot exceed 15. (That is, if there are 11 digits in the pound-report-string, there cannot be more than 4 digits in the fractional-pence-field.)

* The character £ is the sterling equivalent of $.

* Termination is controlled by the pound-separator-string.

Pound-Separator-String - This string is optional. It may include one character, or any combination of the following characters:

    B : / . (period or decimal point)

Editing of the separator characters is dependent upon the use of C or D as delimiters.

The Delimiter Characters - The delimiter characters C and D are required. They primarily serve to indicate the end of the pounds and shillings portions of the picture. In addition, they serve to indicate the type of editing to be applied to separator characters to the right of the low-order digit (of the pounds and shillings integer portions of the item).

The delimiter character D indicates that separator character(s) to the right of the low-order digit position (of the field delimited) are always to appear; that is, no editing is performed on the separator character(s).

The delimiter character C indicates that if the low-order digit position (of the field delimited) is represented by other than the edit character 9, editing continues through the separator character(s).

The delimiter characters C and D are used for editing purposes only. They do not take up space in the printed result.

The following examples show the editing performed when a value of zero is moved to a sterling report item.

        **/CZ9s/D99d

would result in

        ***b0s/00d

whereas, if the picture were

        **/DZ9s/D99d

the result would be

        **/b0s/00d

   The delimiter C is equivalent to D when the low-order digit position is represented by a 9. That is, the following two pictures are equivalent:

        ZZ9/DZ9/D99
        ZZ9/CZ9/C99

   The delimiters used for the pounds and shillings portion of the picture need not be the same.

Note: Although the pound-report-string and the pound-separator-string are optional, a delimiter character (either C or D) must be present; thus, when programming for shillings and pence only, the PICTURE clause must begin PICTURE IS C (or D).

Shilling-Report-String - This is a required two-character field. It is made up of the following characters:

   9 8 X *

The valid combinations of these characters are:

   99 Z9 ZZ Z8 *9 ** *8

The 8 is an edit character and is treated as a 9. However, if the digits to the left of the edit character 8 are zeros, the 8 is treated as the character that precedes it (either Z or *).

Shilling-Separator-String - This string is optional. It may include one character, or any combination of the following characters:

   B : / s . (period or decimal point)

   Editing of the shilling-separator characters is dependent upon the use of C or D as delimiters.

Pence-Report-String - This field is made up of two parts: a required whole-pence field, and an optional fractional-pence field.

   The required whole-pence field is a two-character field, made up of the following symbols:

        9 8 Z *

Valid combinations of these characters are:

   99 Z9 ZZ Z8 *9 ** *8

The function of the editing character 8 is the same as in the shilling-report-string.

The optional fractional-pence field is indicated by a decimal point followed by one or more 9's. It is used to specify fractional pence in decimal form.

The total number of digits in the fractional-pence field plus the pound-report-string cannot exceed 15.

Pence-Separator-String - This string is optional and may consist of one or both of the following characters:

    d . (period or decimal point)

If both characters are used, they must be used in the order shown above.

Sign-Field - This field is optional and may consist of:

• optionally, one or more blanks (B), followed by

• one of the following one- or two-character combinations:

    + - CR DB

Sterling Report editing applications are shown in Table 25.

Table 26. Sterling Currency Editing Applications

Picture	Numeric Value (in pence)	Sterling Equivalent £ s d	Printed Result
£££ /D99s/D99d	3068	12 15 08	£12/15s/08d
£££ /D99s/D99d	0668	2 15 08	£2/15s/08d
£££ /D99s/D99d	0188	0 15 08	/15s/08d
£££ :C99s:D99d	0188	0 15 08	15s/08d
ZZZ/DZZs/DZZd	0000	0 00 00	/ s/ d
ZZZ/CZZs/DZZd	0000	0 00 00	s/ d
£BD99sBD99.9d	080.5	0 06 08.5	06s 08.5d
***/C**D/C**.99d	1040.12	4 06 08.12	**4/*6s/*8.12d
***:C**s:C**.99d	080.12	0 06 08.12	*****6s:*8.12d
***/D**s/D**.99d	00001.23	0 00 01.23	***/**s/*1.23d
£££ /D*9s/D**.99d	00961.23	4 00 01.23	£4/*0s/*1.23d
£**/D*9s/D**.99d	00961.23	4 00 01.23	£*4/*0s/*1.23d
£**/D*9s/D**.99d	00001.23	0 00 01.23	£**/*0s/*1.23d
£££ /D99s/D99dCR	-3068	12 15 08	£12/15s/08dCR

A sterling report PICTURE may have a BLANK WHEN ZERO clause associated with it specifying that the item described is filled with spaces whenever the value of the item is zero.

If the VALUE clause is specified for a Sterling Report item, the literal must be alphanumeric. The VALUE clause is treated exactly as it is specified, with no editing performed.

The maximum length of a sterling report item is 127 characters.

PROCEDURE DIVISION CONSIDERATIONS

The MOVE, DISPLAY, ACCEPT, EXAMINE, and TRANSFORM statements, arithmetic statements, and relation tests may be written containing identifiers that represent sterling items.

If a nonsterling value is moved into a Sterling report item, the compiler treats the value as pence, and converts it to pounds/shillings/pence.

Sterling items are not considered to be integers and should not be used where an integer is required.

INTERNATIONAL CONSIDERATIONS

1. The functions of the period and the comma may be exchanged in the PICTURE character-string and in numeric literals by writing the clause DECIMAL-POINT IS COMMA in the SPECIAL-NAMES paragraph of the Environment Division.

2. The PICTURE of report items may terminate with the currency symbol in cases where the graphic $ is supplanted by a particular national currency symbol, through use of the CURRENCY SIGN IS literal clause in the SPECIAL-NAMES paragraph of the Environment Division.
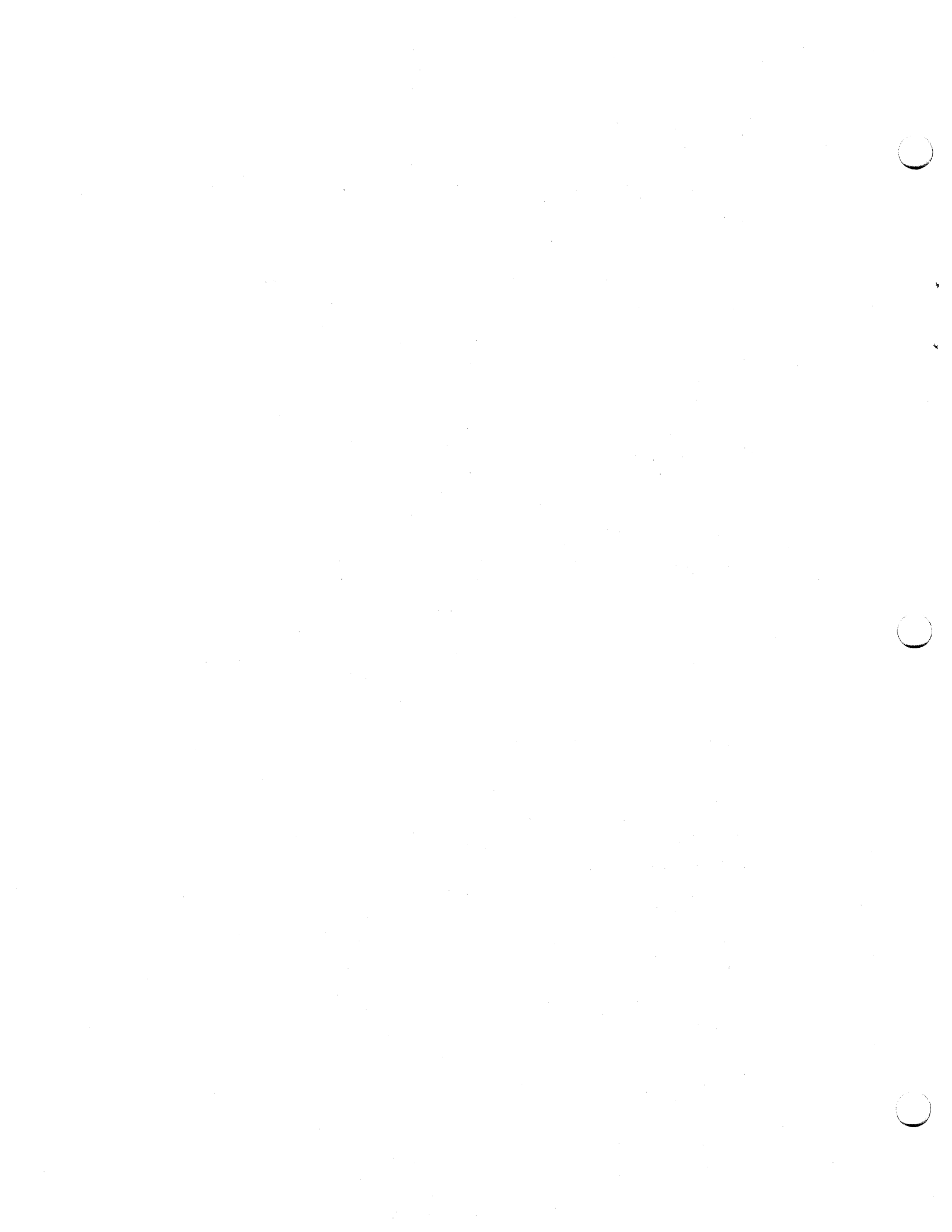
- APPENDIXES

    A: Intermediate Results

    B: Sample Programs

    C: American National Standard COBOL Formats and Reserved Words

    D: File-Processing Techniques and Applicable Statements and Clauses

    E: ASCII Considerations (Version 3)

    F: Symbolic Debugging Feature (Version 3)

    G: 3525 Combined Function Processing (Version 3)

- IBM American National Standard COBOL GLOSSARY
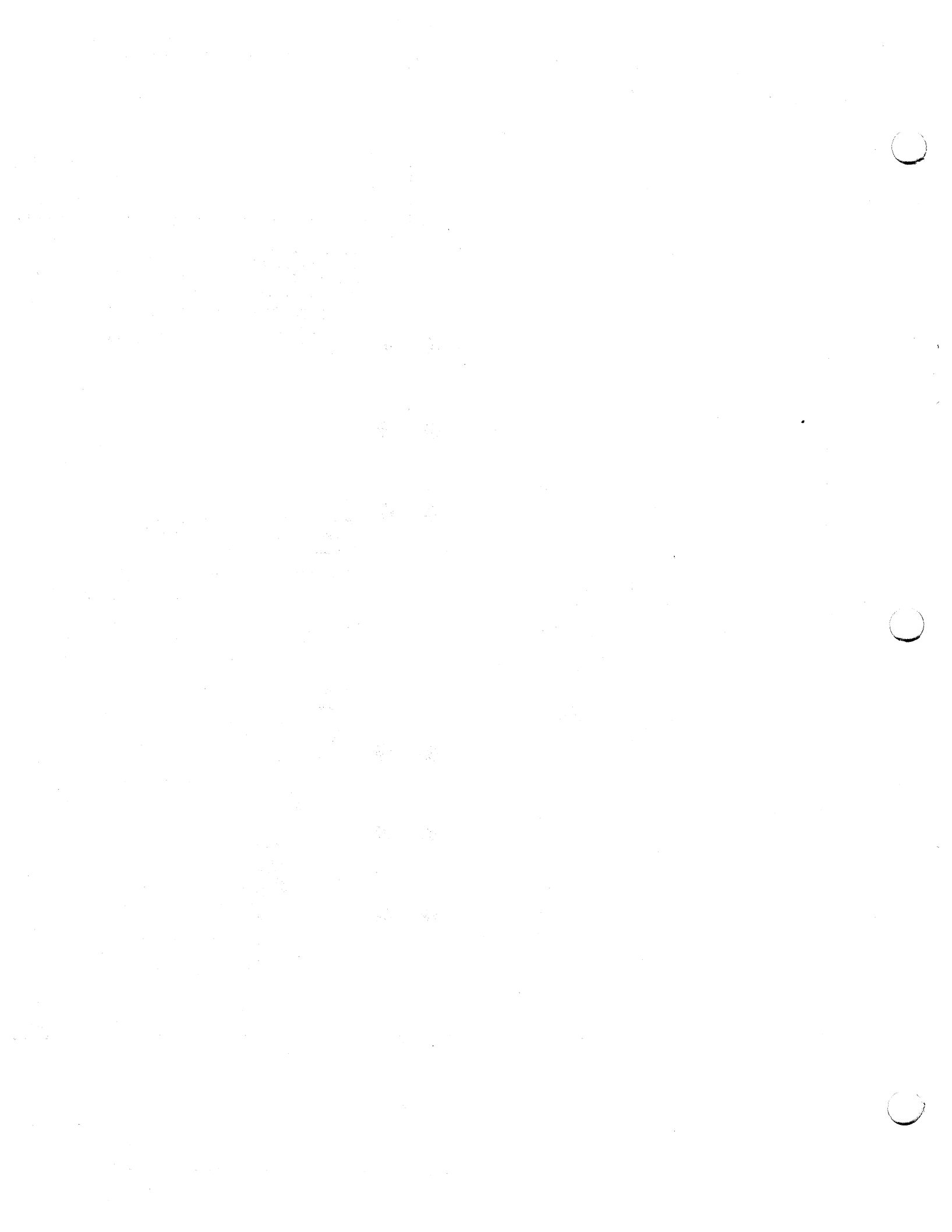
This appendix discusses the conceptual compiler algorithms for determining the number of integer and decimal places reserved for intermediate results.   The following abbreviations are used:

i--number of integer places carried for an intermediate result

d--number of decimal places carried for an intermediate result

dmax--maximum number of decimal places defined for any operand (except for data-names used as exponents) in a particular statement

op1--first operand in a generated arithmetic statement

op2--second operand in a generated arithmetic statement

d1,d2--number of decimal places defined for op1 or op2, respectively

ir--intermediate result field obtained from the execution of a generated arithmetic statement or operation.   Ir1, ir2, etc., represent successive intermediate results.   These intermediate results are generated either in registers or in storage locations.   Successive intermediate results may have the same location.

In the case of an arithmetic statement containing only a single pair of operands, no intermediate results are generated, except when the TRUNC option is specified for COMPUTATIONAL items.   Intermediate results are possible in the following cases:

1.   In an ADD or SUBTRACT statement containing multiple operands immediately following the verb.

2.   In a COMPUTE statement for a series of arithmetic operations.

3.   In arithmetic expressions contained in IF or PERFORM statements.

In such cases, the compiler treats the statement as a succession of operations.   For example, the following statement:

    COMPUTE Y = A + B * C - D / E + F ** G

is replaced by

    ** F   BY G   yielding ir1
    MULTIPLY B   BY C   yielding ir2
    DIVIDE E   INTO D   yielding ir3
    ADD A   TO ir2   yielding ir4
    SUBTRACT ir3   FROM ir4   yielding ir5
    ADD ir5   TO ir1   yielding Y

COMPILER CALCULATION OF INTERMEDIATE RESULTS

The number of integer places in an ir is calculated as follows:

• The compiler first determines the maximum value the ir can contain by performing the statement in which the ir occurs.

    1.   If an operand in this statement is a data-name, the value used for the data-name is equal to the numerical value of the PICTURE for data-name (e.g., PICTURE 9V99 has value 9.99).

2. If an operand is a literal, the literal's actual value is used.

3. If an operand is an intermediate result, the value determined for the intermediate result in a previous arithmetic operation is used.

4. If the operation is division:

   a. If op2 is a data-name, the value used for op2 is the minumum nonzero value of the digit in the PICTURE for the data-name (e.g., PICTURE 9V99 has the value 0.01).

   b. If op2 is an intermediate result, the intermediate result is treated as though it had a PICTURE, and the minimum nonzero value of the digits in this PICTURE is used.

- When the maximum value of the ir is determined by the above procedures, i is set equal to the number of integers in the maximum value.

- The number of decimal places contained in an ir is calculated as:

Operation	Decimal Places
+ or -	d1 or d2, whichever is greater
*	d1 + d2
/	d1 - d2 or dmax, whichever is greater
**	dmax if op2 is nonintegral or a data-name; d1 * op2 if op2 is an integral literal

Table 27 indicates the action of the compiler when handling intermediate results.

Table 27. Compiler Action on Intermediate Result

Value of i + d	Value of d	Value of i + dmax	Action Taken
<30	Any value	Any value	i integer and d decimal places are carried for ir
=30			
>30	<dmax	Any value	30 - d integer and d decimal places are carried for ir
	=dmax		
	>dmax	<30	i integer and 30 - i decimal places are carried for ir
		=30	
		>30	30 - dmax integer and dmax decimal places are carried for ir

The three programs in this appendix illustrate several methods of accessing mass storage files.   The three programs are:

1.   CREATION OF A DIRECT FILE

2.   CREATION OF AN INDEXED FILE

3.   RANDOM RETRIEVAL AND UPDATING OF AN INDEXED FILE

CREATION OF A DIRECT FILE

    This program creates a file with direct organization through use of
an ACTUAL KEY ¹- using actual track addressing. The ACTUAL KEY has two
components:

1. A track identifier, which is the actual track address, calculated
    through a simple remainder randomizing technique. A field from the
    input record (CD-ITEM-CODE) is converted to a track address
    (TRACK1). CD-ITEM-CODE is divided by 19; the remainder is placed
    in TRACK1. This gives a valid track address ranging in value from
    0 to 18. The program assumes that the file must begin on cylinder
    102; therefore, a displacement value of 1020 is added to assure
    that the CC field will contain the value of decimal 102.

2. A unique record identifier, in this case TRACK-NAME, that contains
    the CD-ITEM-NAME of the input record.

    The UPDATING program in the introduction updates the file that this
program creates.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.   CREATEDF.
REMARKS.   ILLUSTRATE CREATION OF A DIRECT FILE.  THE FILE MAY BE
    USED AS THE MASTER FILE FOR THE SAMPLE PROGRAM ILLUSTRATED
    IN THE INTRODUCTION TO THIS MANUAL.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.   IBM-360-F50.
OBJECT-COMPUTER.   IBM-360-F50.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT DA-FILE ASSIGN TO SYS015-DA-2311-A-MASTER
        ACCESS IS RANDOM
        ACTUAL KEY IS FILEKEY.
    SELECT CARD-FILE ASSIGN TO SYS007-UR-2540R-S.
DATA DIVISION.
FILE SECTION.
FD  DA-FILE
    DATA RECORD IS DISK
    LABEL RECORDS ARE STANDARD.
01  DISK.
    02   DISK-ITEM-CODE      PICTURE X(3).
    02   DISK-ITEM-NAME      PICTURE X(29).
    02   DISK-STOCK-ON-HAND  PICTURE S9(6) USAGE COMP SYNC.
    02   DISK-UNIT-PRICE     PICTURE S999V99 USAGE COMP SYNC.
    02   DISK-STOCK-VALUE    PICTURE S9(9)V99 USAGE COMP SYNC.
    02   DISK-ORDER-POINT    PICTURE S9(3) USAGE COMP SYNC.
FD  CARD-FILE
    LABEL RECORDS ARE OMITTED
    DATA RECORD IS CARDS.
01  CARDS.
    02   CD-ITEM-CODE        PICTURE X(3).
    02   CD-ITEM-NAME        PICTURE X(29).
    02   CD-STOCK-ON-HAND    PICTURE S9(6).
    02   CD-UNIT-PRICE       PICTURE S999V99.
    02   CD-STOCK-VALUE      PICTURE S9(9)V99.
    02   CD-ORDER-POINT      PICTURE S9(3).
    02   FILLER              PICTURE X(23).
```

```
      WORKING-STORAGE SECTION.
      77   SAVE                    PICTURE S9(5) USAGE COMP SYNC.
      77   QUOTIENT                PICTURE S9(4) USAGE COMP SYNC.
      01   TRACK1                  PICTURE 9999.
      01   TRACK2 REDEFINES TRACK1.
           02   CYL                PICTURE 999.
           02   HEAD               PICTURE 9.
      01   KEY-1.
           02   M                  PICTURE S999 COMP SYNC VALUE ZEROES.
           02   BB                 PICTURE S9   COMP SYNC VALUE ZERO.
           02   CC                 PICTURE S999 COMP SYNC.
           02   HH                 PICTURE S9   COMP SYNC.
           02   R                  PICTURE X    VALUE LOW-VALUE.
           02   TRACK-NAME         PICTURE X(29).
      01   THE-KEY REDEFINES KEY-1.
           02   FILLER             PICTURE X.
           02   FILEKEY            PICTURE X(37).
      PROCEDURE DIVISION.
      BEGIN.
          OPEN INPUT CARD-FILE.
          OPEN OUTPUT DA-FILE.
      PARA-1.
          READ CARD-FILE AT END GO TO END-JOB.
          MOVE CD-ITEM-CODE TO SAVE.
          DIVIDE 19 INTO SAVE GIVING QUOTIENT
              REMAINDER TRACK1.
          ADD 1020 TO TRACK1.
          MOVE CD-ITEM-NAME TO TRACK-NAME.
          MOVE CYL TO CC.
          MOVE HEAD TO HH.
          MOVE CD-ITEM-CODE TO DISK-ITEM-CODE.
          MOVE CD-ITEM-NAME TO DISK-ITEM-NAME.
          MOVE CD-STOCK-ON-HAND TO DISK-STOCK-ON-HAND.
          MOVE CD-UNIT-PRICE TO DISK-UNIT-PRICE.
          MOVE CD-STOCK-VALUE TO DISK-STOCK-VALUE.
          MOVE CD-ORDER-POINT TO DISK-ORDER-POINT.
      WR.
          WRITE DISK INVALID KEY GO TO ERROR-ROUTINE.
          GO TO PARA-1.
      END-JOB.
          CLOSE CARD-FILE DA-FILE.
          DISPLAY "END OF JOB".
          STOP RUN.
      ERROR-ROUTINE.
          DISPLAY "UNABLE TO WRITE RECORD".
          DISPLAY TRACK-NAME.
          GO TO PARA-1.
```

CREATION OF AN INDEXED FILE

This program creates an indexed file. These records are presented in ascending sequence by RECORD KEY. The APPLY clause builds the master index.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CREATEIS.
REMARKS. ILLUSTRATE CREATION OF INDEXED SEQUENTIAL FILE.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-360-F50.
OBJECT-COMPUTER. IBM-360-F50.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT IS-FILE ASSIGN TO SYS015-DA-2311-I-MASTER
        ACCESS IS SEQUENTIAL
        RECORD KEY IS REC-ID.
    SELECT CARD-FILE ASSIGN TO SYS007-UR-2540R-S
        RESERVE 1 ALTERNATE AREA.
I-O-CONTROL.
    APPLY MASTER-INDEX TO 2311 ON IS-FILE.
DATA DIVISION.
FILE SECTION.
FD  IS-FILE
    BLOCK CONTAINS 5 RECORDS
    RECORDING MODE IS F
    LABEL RECORDS ARE STANDARD
    DATA RECORD IS DISK.
01  DISK.
    02   DELETE-CODE      PICTURE X.
    02   REC-ID           PICTURE 9(10).
    02   DISK-FLD1        PICTURE X(10).
    02   DISK-NAME        PICTURE X(20).
    02   DISK-BAL         PICTURE 99999V99.
    02   FILLER           PICTURE X(52).
FD  CARD-FILE
    RECORDING MODE IS F
    LABEL RECORDS ARE OMITTED
    DATA RECORD IS CARDS.
01  CARDS.
    02   KEY-ID           PICTURE 9(10).
    02   CD-NAME          PICTURE X(20).
    02   CD-BAL           PICTURE 99999V99.
    02   FILLER           PICTURE X(43).
```

```
PROCEDURE DIVISION.
BEGIN.
    OPEN INPUT CARD-FILE.
    OPEN OUTPUT IS-FILE.
PARA-1.
    READ CARD-FILE AT END GO TO END-JOB.
    MOVE LOW-VALUE TO DELETE-CODE.
    MOVE KEY-ID TO REC-ID.
    MOVE CD-NAME TO DISK-NAME.
    MOVE CD-BAL TO DISK-BAL.
    WRITE DISK INVALID KEY GO TO ERR.
    GO TO PARA-1.
ERR.
    DISPLAY "DUPLICATE OR SEQ-ERR" UPON CONSOLE.
    DISPLAY KEY-ID UPON CONSOLE.
    GO TO PARA-1.
END-JOB.
    CLOSE CARD-FILE IS-FILE.
    DISPLAY "END OF JOB" UPON CONSOLE.
    STOP RUN.
```

This program randomly updates an existing indexed file. The READ
IS-FILE statement causes a search of indexes for an equal compare
between the NOMINAL KEY obtained from the input record and the RECORD
KEY of the I-O file. If an equal compare occurs, the record is updated,
and the details of this update are printed. If a matching record is not
found, the invalid key branch is taken.


```
IDENTIFICATION DIVISION.
PROGRAM-ID. RANDOMIS.
REMARKS. ILLUSTRATE RANDOM  RETRIEVAL FROM IS-FILE.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-360-F50.
OBJECT-COMPUTER. IBM-360-F50.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT IS-FILE ASSIGN TO SYS015-DA-2311-I-MASTER
        ACCESS IS RANDOM
        NOMINAL KEY IS KEY-ID
        RECORD KEY IS REC-ID.
    SELECT CARD-FILE ASSIGN TO SYS007-UR-2540R-S
        RESERVE 1 ALTERNATE AREA.
    SELECT PRINT-FILE ASSIGN TO SYS008-UT-2400-S-PROUT
        RESERVE NO ALTERNATE AREAS.
I-O-CONTROL.
    APPLY MASTER-INDEX TO 2311 ON IS-FILE.
    RERUN ON SYS002-UT-2400-S-CKPT EVERY 10000 RECORDS
        OF IS-FILE.
DATA DIVISION.
FILE SECTION.
FD   IS-FILE
    BLOCK CONTAINS 5 RECORDS
    RECORD CONTAINS 100 CHARACTERS
    LABEL RECORDS ARE STANDARD
    RECORDING MODE IS F
    DATA RECORD IS DISK.
01   DISK.
    02   DELETE-CODE       PICTURE X.
    02   REC-ID            PICTURE 9(10).
    02   DISK-FLD1         PICTURE X(10).
    02   DISK-NAME         PICTURE X(20).
    02   DISK-BAL          PICTURE 99999V99.
    02   FILLER            PICTURE X(52).
FD   CARD-FILE
    RECORDING MODE IS F
    LABEL RECORDS ARE OMITTED
    DATA RECORD IS CARDS.
01   CARDS.
    02   KEY-IDA           PICTURE 9(10).
    02   CD-NAME           PICTURE X(20).
    02   CD-AMT            PICTURE 99999V99.
    02   FILLER            PICTURE X(43).
FD   PRINT-FILE
    RECORDING MODE IS F
    LABEL RECORDS ARE STANDARD
    DATA RECORD IS PRINTER.
```

```
01   PRINTER.
     02   FORMSC           PICTURE X.
     02   PRINT-ID         PICTURE X(10).
     02   FILLER           PICTURE X(10).
     02   PRINT-NAME       PICTURE X(20).
     02   FILLER           PICTURE X(10).
     02   PRINT-BAL        PICTURE $ZZZ,999.99-.
     02   FILLER           PICTURE X(10).
     02   PRINT-AMT        PICTURE $ZZZ,ZZZ.99-.
     02   FILLER           PICTURE X(10).
     02   PRINT-NEW-BAL    PICTURE $ZZZ,ZZZ.99-.
WORKING-STORAGE SECTION.
77   KEY-ID               PICTURE 9(10).
PROCEDURE DIVISION.
BEGIN.
     OPEN INPUT CARD-FILE.
     OPEN OUTPUT PRINT-FILE.
     OPEN I-O IS-FILE.
PARA-1.
     MOVE SPACES TO PRINTER.
     READ CARD-FILE AT END GO TO END-JOB.
     MOVE KEY-IDA TO KEY-ID.
     READ IS-FILE INVALID KEY GO TO NO-RECORD.
     MOVE REC-ID TO PRINT-ID.
     MOVE DISK-NAME TO PRINT-NAME.
     MOVE DISK-BAL TO PRINT-BAL.
     MOVE CD-AMT TO PRINT-AMT.
     ADD CD-AMT TO DISK-BAL.
     MOVE DISK-BAL TO PRINT-NEW-BAL.
     REWRITE DISK INVALID KEY GOTO NO-RECORD.
     WRITE PRINTER AFTER POSITIONING 2 LINES.
     GO TO PARA-1.
NO-RECORD.
     DISPLAY 'NO RECORD FOUND' UPON CONSOLE.
     DISPLAY KEY-ID UPON CONSOLE.
     GO TO PARA-1.
END-JOB.
     CLOSE CARD-FILE PRINT-FILE IS-FILE.
     DISPLAY 'END OF JOB' UPON CONSOLE.
     STOP RUN.
```

The Formats and Reserved Words in this appendix have been printed in a specially reduced size with pages numbered in sequence to make up a pocket-sized reference booklet for use while coding IBM American National Standard COBOL programs. Although most readers may prefer to retain this reference material within the manual, the booklet can be prepared as follows:

- Cut along trim lines.

- Place sheets so that page numbers at lower right-hand corner are in ascending order in odd-number progression (i.e., 1, 3, 5, etc.); lower left-hand page numbers will then be in descending order in even-number progression (i.e., 16, 14, 12, etc.).

- Fold trimmed sheets after collating.

- Staple along fold if desired.

- Punch for six-hole binder.

TRIM HERE

Symbols Allowed in the PICTURE Clause

Symbol	Meaning
A	Alphabetic character or space
B	Space insertion character
P	Decimal scaling position (not counted in size of data item)
S	Operational sign (not counted in size of data item)
V	Assumed decimal point (not counted in size of data item)
X	Alphanumeric character (any from the EBCDIC set)
Z	Zero suppression character
9	Numeric character
0	Zero insertion character
,	Comma insertion character
.	Decimal point or period editing control character
+	Plus sign insertion editing control character
−	Minus sign editing control character
CR	Credit editing control characters
DB	Debit editing control characters
*	Check protect insertion character
$	Currency sign insertion character

IBM System/360 Reference Data

IBM American National Standard COBOL

Disk Operating System

Appendix C: IBM American National Standard COBOL Format Summary and Reserved Words

The general format of a COBOL program is illustrated in these format summaries. Included within the general format is the specific format for each valid COBOL statement. All clauses are shown as though they were required by the COBOL source program, although within a given context many are optional. Repetition of clauses has not been indicated. Several formats are included under special headings, which are different from, or additions to, the general format. Under these special headings are included formats peculiar to the following COBOL features: Sort, Report Writer, Table Handling, Segmentation, Source Program Library Facility, Debugging Language, Format Control of the Source Program Listing, and Sterling Currency. Each of these features is explained within a special chapter of this publication — *IBM System/360 Disk Operating System: American National Standard COBOL*, Form GC28-6394.

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza
New York, New York 10017
(International)

Printed in U.S.A.   GC28-6394

TRIM HERE

IDENTIFICATION DIVISION — BASIC FORMATS

{IDENTIFICATION DIVISION.}
{ID DIVISION.}

PROGRAM-ID. program-name.
AUTHOR. [comment-entry] ...
INSTALLATION. [comment-entry] ...
DATE-WRITTEN. [comment-entry] ...
DATE-COMPILED. [comment-entry] ...
SECURITY. [comment-entry] ...
REMARKS. [comment-entry] ...

ENVIRONMENT DIVISION — BASIC FORMATS

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. computer-name.

OBJECT-COMPUTER. computer-name [MEMORY SIZE integer {WORDS / CHARACTERS / MODULES}]

[SEGMENT-LIMIT IS priority-number].

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

[function-name-2 [IS mnemonic-name]

{ON STATUS IS condition-name-1 [OFF STATUS IS condition-name-2]}
{OFF STATUS IS condition-name-2 [ON STATUS IS condition-name-1]}] ...

[CURRENCY SIGN IS literal]

[DECIMAL-POINT IS COMMA].

INPUT-OUTPUT SECTION.

FILE-CONTROL.

{SELECT [OPTIONAL] file-name

ASSIGN TO [integer] system-name-1 [system-name-2] ...

[FOR MULTIPLE {REEL / UNIT}]

RESERVE {NO / integer} ALTERNATE [AREA / AREAS]

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

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

ACCESS MODE IS {SEQUENTIAL / RANDOM}

PROCESSING MODE IS SEQUENTIAL

ACTUAL KEY IS data-name

NOMINAL KEY IS data-name

RECORD KEY IS data-name

TRACK-AREA IS integer CHARACTERS.} ...

I-O-CONTROL.

RERUN ON system-name EVERY integer RECORDS OF file-name

SAME [SORT / RECORD] AREA FOR file-name-1 {file-name-2} ...

MULTIPLE FILE TAPE CONTAINS file-name-1 [POSITION integer-1]

[file-name-2 [POSITION integer-2]] ...

Structure of the First Eight Bytes of ACTUAL KEY When Using Actual Track Addressing

	Pack	Cell		Cylinder		Head		Record
Byte	M	B	B	C	C	H	H	R
Device	0	1	2	3	4	5	6	7
2311	0-221	0	0	0	0-199	0	0-9	0-255
2314	0-221	0	0	0	0-199	0	0-19	0-255
2321	0-221	0	0-9	0-19	0-9	0-4	0-19	0-255

Structure of system-name in the ASSIGN clause

SYSnnn-class-device-organization [-name]

nnn -- a three-digit number between 000 and 221

class -- a two-character field that represents device class.
Allowable combinations are:

DA   mass storage
UT   utility
UR   unit record

device -- a four- or five-digit field that represents device
number. Allowable numbers for each device class
are:

DA   2311, 2321, 2314
UT   2400, 2311, 2314, 2321
UR   1442R, 1442P, 1403, 1404 (continuous forms
only), 1443, 2501, 2520R, 2520P, 2540R,
2540P

Program Product Information -- Version 3

The following additional device class numbers are valid:

DA   2319, 3330
UT   2319, 3330, 3410, 3420
UR   3211, 3505, 3525R, 3525P, 3525W, 3525M

organization -- a one-character field that specifies file
organization. Allowable characters are:

S   for standard sequential files
A   for direct files -- actual track addressing
D   for direct files -- relative track addressing
U   for direct files using REWRITE statement*
-- actual track addressing
W   for direct files using REWRITE statement*
-- relative track addressing
I   for indexed files

* When the file is opened INPUT or OUTPUT,
U and W are equivalent to A and D.

Program Product Information -- Version 3

The following additional organization characters are valid:

O   for 3505 OMR reading
S [R] for 3505 and 3525R sequential reading*
T   for 3525 punch-and-interpret files
V [R] for 3525 read/print associated files*
X [R] for 3525 read/punch/print associated files*
Y [R] for 3525 read/punch associated files*
Z   for 3525 punch/print associated files
----
*   the optional R field denotes RCE reading, and
is valid only for the 3505 and 3525R devices.

name -- a one- to seven-character field that specifies the
external-name by which the file is known to the
system.

18

APPLY WRITE-ONLY ON file-name-1 [file-name-2] ...
APPLY EXTENDED-SEARCH ON file-name-1 [file-name-2] ...
APPLY WRITE-VERIFY ON file-name-1 [file-name-2] ...
APPLY CYL-OVERFLOW OF integer TRACKS ON file-name-1 [file-name-2] ...
APPLY {MASTER-INDEX / CYL-INDEX} TO device-number ON file-name-1 [file-name-2] ...
APPLY CORE-INDEX TO data-name ON file-name-1 [file-name-2] ....
NOTE: Format 2 of the RERUN Clause (for Sort files) is included with Formats for the Sort feature.

DATA DIVISION — BASIC FORMATS

DATA DIVISION.

FILE SECTION.

FD file-name

BLOCK CONTAINS [integer-1 TO] integer-2 {CHARACTERS / RECORDS}

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

RECORDING MODE IS mode

LABEL {RECORD IS / RECORDS ARE} {OMITTED / STANDARD / data-name-1 [data-name-2] ...}

VALUE OF data-name-1 IS {data-name-2 / literal-1} [data-name-3 IS ... {data-name-4 / literal-2}]

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

NOTE: The Format of the REPORT Clause is included with Formats for the REPORT writer feature.

01-49 {data-name-1 / FILLER}

REDEFINES data-name-2

BLANK WHEN ZERO

{JUSTIFIED / JUST} RIGHT

{PICTURE / PIC} IS character string

[SIGN IS] {LEADING / TRAILING} [SEPARATE CHARACTER] (Version 3)

{SYNCHRONIZED / SYNC} {LEFT / RIGHT}

[USAGE IS] { INDEX / DISPLAY / COMPUTATIONAL / COMP / COMPUTATIONAL-1 / COMP-1 / COMPUTATIONAL-2 / COMP-2 / COMPUTATIONAL-3 / COMP-3 / DISPLAY-ST }

88 condition-name {VALUE IS / VALUES ARE} literal-1 [THRU literal-2]
[literal-3 [THRU literal-4]] ....

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

NOTE: Formats of the OCCURS Clause are included with Formats for the TABLE HANDLING feature.

3

FOLD

TRIM HERE

WORKING-STORAGE SECTION.

77 data-name-1

01-49 {data-name-1 / FILLER}

REDEFINES data-name-2

BLANK WHEN ZERO

{JUSTIFIED / JUST} RIGHT

{PICTURE / PIC} IS character string

[SIGN IS] {LEADING / TRAILING} [SEPARATE CHARACTER] (Version 3)

{SYNCHRONIZED / SYNC} {LEFT / RIGHT}

[USAGE IS] {INDEX / DISPLAY / COMPUTATIONAL / COMP / COMPUTATIONAL-1 / COMP-1 / COMPUTATIONAL-2 / COMP-2 / COMPUTATIONAL-3 / COMP-3 / DISPLAY-ST}

VALUE IS literal .

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

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

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

NOTE: Formats of the OCCURS Clause are included with Formats for the TABLE HANDLING feature.

LINKAGE SECTION.

77 data-name-1

01-49 {data-name-1 / FILLER}

REDEFINES data-name-2

BLANK WHEN ZERO

{JUSTIFIED / JUST} RIGHT

{PICTURE / PIC} IS character string

[SIGN IS] {LEADING / TRAILING} [SEPARATE CHARACTER] (Version 3)

{SYNCHRONIZED / SYNC} {LEFT / RIGHT}

[USAGE IS] {INDEX / DISPLAY / COMPUTATIONAL / COMP / COMPUTATIONAL-1 / COMP-1 / COMPUTATIONAL-2 / COMP-2 / COMPUTATIONAL-3 / COMP-3 / DISPLAY-ST}

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

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

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

NOTE: Formats of the OCCURS Clause are included with Formats for the TABLE HANDLING feature.

4

PLUS
(ca) POINTER
POSITION
(xac) POSITIONING
POSITIVE
(xac) PRINT-SWITCH
(ca) PRIORITY
PROCEDURE
PROCEED
(ca) PROCESS
PROCESSING
(xa) PROGRAM
PROGRAM-ID
(ca) QUEUE
QUOTE
QUOTES
RANDOM
RD
READ
(xac) READY
(ca) RECEIVE
RECORD
(spn) RECORD-OVERFLOW
(xa) RECORDING
RECORDS
REDEFINES
(ca) REFERENCES
REEL
RELEASE
REMAINDER
REMARKS
RENAMES
(spn) REORG-CRITERIA
REPLACING
REPORT
REPORTING
REPORTS
(spn) REREAD
RERUN
RESERVE
RESET
RETURN
(spn) RETURN-CODE
REVERSED
REWIND
(xac) REWRITE
RF
RH
RIGHT
ROUNDED
RUN
(ca) SA
SAME
SD
SEARCH
SECTION
SECURITY
SEEK
(ca) SEGMENT
SEGMENT-LIMIT
SELECT
(ca) SELECTED
(ca) SEND
SENTENCE
(xa) SEPARATE
SEQUENTIAL
SET
SIGN
SIZE
(xac) SKIP1
(xac) SKIP2
(xac) SKIP3
SORT
(xac) SORT-CORE-SIZE
(xac) SORT-FILE-SIZE
(xac) SORT-MODE-SIZE
(xac) SORT-RETURN
SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD

(xac) START
STATUS
STOP
(ca) STRING
(ca) SUB-QUEUE-1
(ca) SUB-QUEUE-2
(ca) SUB-QUEUE-3
SUBTRACT
SUM
(ca) SUPERVISOR
(xac) SUPPRESS
(ca) SUSPEND
(ca) SYMBOLIC
SYNC
SYNCHRONIZED
(spn) SYSIN
(sp) SYSIPT
(sp) SYSLST
(spn) SYSOUT
(sp) SYSPCH
(sp) SYSPUNCH
(sp) S01
(sp) S02
(ca) TABLE
TALLY
TALLYING
TAPE
(ca) TERMINAL
TERMINATE
(ca) TEXT
THAN
(xac) THEN
THROUGH
THRU
(ca) TIME
(xac) TIME-OF-DAY
TIMES
TO
(spn) TOTALED
(spn) TOTALING
(xac) TRACE
(spn) TRACK
(xac) TRACK-AREA
(spn) TRACK-LIMIT
(xac) TRACKS
(xa) TRAILING
(xac) TRANSFORM
TYPE
(ca) UNEQUAL
UNIT
(ca) UNSTRING
UNTIL
UP
UPON
(ca) UPPER-BOUND
(ca) UPPER-BOUNDS
(sp) UPSI-0
(sp) UPSI-1
(sp) UPSI-2
(sp) UPSI-3
(sp) UPSI-4
(sp) UPSI-5
(sp) UPSI-6
(sp) UPSI-7
USAGE
USE
USING
VALUE
VALUES
VARYING
WHEN
WITH
WORDS
WORKING-STORAGE
WRITE
(xac) WRITE-ONLY
(xac) WRITE-VERIFY
ZERO
ZEROES
ZEROS

17

TRIM HERE

FOLD

Column 1:

DATE-WRITTEN
(ca) DAY
DE
(xac) DEBUG
(ca) DEBUG-CONTENTS
(ca) DEBUG-LINE
(ca) DEBUG-NAME
(ca) DEBUG-SUB-n
(ca) DEBUGGING
DECIMAL-POINT
DECLARATIVES
(xac) DELETE
(ca) DELIMITED
(ca) DELIMITER
DEPENDING
(ca) DEPTH
DESCENDING
(ca) DESTINATION
DETAIL
(ca) DISABLE
(spn) DISP
DISPLAY
(xac) DISPLAY-ST
(ca) DISPLAY-n
DIVIDE
DIVISION
DOWN
(xac) DYNAMIC

(xac) EJECT
ELSE
(ca) EMI
(ca) ENABLE
END
(xa) END-OF-PAGE
ENDING
ENTER
(xac) ENTRY
ENVIRONMENT
(xa) EOP
EQUAL
(ca) EQUALS
ERROR
(ca) ESI
(ca) ETI
EVERY
EXAMINE
(ca) EXCEEDS
(xac) EXHIBIT
EXIT
(xac) EXTENDED-SEARCH

FD
FILE
FILE-CONTROL
FILE-LIMIT
FILE-LIMITS
FILLER
FINAL
FIRST
FOOTING
FOR
(xac) FREE
FROM

GENERATE
GIVING
GO
(xac) GOBACK
GREATER
GROUP

HEADING
HIGH-VALUE
HIGH-VALUES
(ca) HOLD

I-O
I-O-CONTROL
(xac) ID
IDENTIFICATION
IF
IN
INDEX
(ca) INDEX-n
INDEXED

Column 2:

(ca) INDICATE
INITIAL
INITIATE
INPUT
INPUT-OUTPUT
(xac) INSERT
(ca) INSPECT
INSTALLATION
INTO
INVALID
IS

JUST
JUSTIFIED

KEY
(ca) KEYS
LABEL
(spn) LABEL-RETURN
LAST
LEADING
(spn) LEAVE
LEFT
(ca) LENGTH
LESS
(ca) LIBRARY
LIMIT
LIMITS
(ca) LINAGE
(ca) LINAGE-COUNTER
LINE
LINE-COUNTER
LINES
(xa) LINKAGE
LOCK
LOW-VALUE
LOW-VALUES
(ca) LOWER-BOUND
(ca) LOWER-BOUNDS

(xac) MASTER-INDEX
MEMORY
(ca) MESSAGE
MODE
MODULES
(xac) MORE-LABELS
MOVE
MULTIPLE
MULTIPLY

(xac) NAMED
NEGATIVE
NEXT
NO
(xac) NOMINAL
NOT
NOTE
(xac) NSTD-REELS
NUMBER
NUMERIC
(ca) NUMERIC-EDITED

OBJECT-COMPUTER
(ca) OBJECT-PROGRAM
OCCURS
OF
OFF
(ca) OH
OMITTED
ON
OPEN
OPTIONAL
OR
(xac) OTHERWISE
OUTPUT
(ca) OV
(ca) OVERFLOW

PAGE
PAGE-COUNTER
PERFORM
PF
PH
PIC
PICTURE

16

Right column:

PROCEDURE DIVISION — BASIC FORMATS

$$\left\{\begin{array}{l}\underline{\text{PROCEDURE}}\ \underline{\text{DIVISION}}.\\ \underline{\text{PROCEDURE}}\ \underline{\text{DIVISION}}\ \underline{\text{USING}}\ identifier\text{-}1\ [identifier\text{-}2]\ \ldots\ .\end{array}\right\}$$

ACCEPT Statement

$$\underline{\text{ACCEPT}}\ identifier\ [\text{FROM}\ \left\{\begin{array}{l}\underline{\text{SYSIPT}}\\ \underline{\text{CONSOLE}}\\ mnemonic\text{-}name\end{array}\right\}]$$

ADD Statement

FORMAT 1

$$\underline{\text{ADD}}\ \left\{\begin{array}{l}identifier\text{-}1\\ literal\text{-}1\end{array}\right\}\ \left[\begin{array}{l}identifier\text{-}2\\ literal\text{-}2\end{array}\right]\ \ldots\ \underline{\text{TO}}\ identifier\text{-}m\ [\underline{\text{ROUNDED}}]$$

$$[identifier\text{-}n\ [\underline{\text{ROUNDED}}]]\ \ldots\ [\text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ imperative\text{-}statement]$$

FORMAT 2

$$\underline{\text{ADD}}\ \left\{\begin{array}{l}identifier\text{-}1\\ literal\text{-}1\end{array}\right\}\ \left\{\begin{array}{l}identifier\text{-}2\\ literal\text{-}2\end{array}\right\}\ \left[\begin{array}{l}identifier\text{-}3\\ literal\text{-}3\end{array}\right]\ \ldots$$

$$\underline{\text{GIVING}}\ identifier\text{-}m\ [\underline{\text{ROUNDED}}]\ [\text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ imperative\text{-}statement]$$

FORMAT 3

$$\underline{\text{ADD}}\ \left\{\begin{array}{l}\underline{\text{CORRESPONDING}}\\ \underline{\text{CORR}}\end{array}\right\}\ identifier\text{-}1\ \text{TO}\ identifier\text{-}2\ [\underline{\text{ROUNDED}}]$$

$$[\text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ imperative\text{-}statement]$$

ALTER Statement

$$\underline{\text{ALTER}}\ procedure\text{-}name\text{-}1\ \underline{\text{TO}}\ [\underline{\text{PROCEED}}\ \underline{\text{TO}}]\ procedure\text{-}name\text{-}2$$

$$[procedure\text{-}name\text{-}3\ \underline{\text{TO}}\ [\underline{\text{PROCEED}}\ \underline{\text{TO}}]\ procedure\text{-}name\text{-}4]\ \ldots$$

CALL Statement

$$\underline{\text{CALL}}\ literal\text{-}1\ [\underline{\text{USING}}\ identifier\text{-}1\ [identifier\text{-}2]\ \ldots]$$

CLOSE Statement

$$\underline{\text{CLOSE}}\ file\text{-}name\text{-}1\ \left[\begin{array}{l}\underline{\text{REEL}}\\ \underline{\text{UNIT}}\end{array}\right]\ [\text{WITH}\ \left\{\begin{array}{l}\underline{\text{NO}}\ \underline{\text{REWIND}}\\ \underline{\text{LOCK}}\end{array}\right\}\ ]$$

$$[file\text{-}name\text{-}2\ \left[\begin{array}{l}\underline{\text{REEL}}\\ \underline{\text{UNIT}}\end{array}\right]\ [\text{WITH}\ \left\{\begin{array}{l}\underline{\text{NO}}\ \underline{\text{REWIND}}\\ \underline{\text{LOCK}}\end{array}\right\}\ ]]\ \ldots$$

COMPUTE Statement

$$\underline{\text{COMPUTE}}\ identifier\text{-}1\ [\underline{\text{ROUNDED}}]\ =\ \left\{\begin{array}{l}arithmetic\text{-}expression\\ identifier\text{-}2\\ literal\text{-}1\end{array}\right\}$$

$$[\text{ON}\ \underline{\text{SIZE}}\ \underline{\text{ERROR}}\ imperative\text{-}statement]$$

DECLARATIVE Section

$$\underline{\text{PROCEDURE}}\ \underline{\text{DIVISION}}.$$
$$\underline{\text{DECLARATIVES}}.$$
$$\{section\text{-}name\ \underline{\text{SECTION}}.\ \underline{\text{USE}}\ sentence.$$
$$\{paragraph\text{-}name.\ \{sentence\}\ \ldots\}\ \ldots\}\ \ldots$$
$$\underline{\text{END}}\ \underline{\text{DECLARATIVES}}.$$

DISPLAY Statement

$$\underline{\text{DISPLAY}}\ \left\{\begin{array}{l}identifier\text{-}1\\ literal\text{-}1\end{array}\right\}\ \left[\begin{array}{l}identifier\text{-}2\\ literal\text{-}2\end{array}\right]\ \ldots\ [\underline{\text{UPON}}\ \left\{\begin{array}{l}\underline{\text{CONSOLE}}\\ \underline{\text{SYSPCH}}\\ \underline{\text{SYSPUNCH}}\\ \underline{\text{SYSLST}}\\ mnemonic\text{-}name\end{array}\right\}\ ]$$

DIVIDE Statement

FORMAT 1

DIVIDE $\begin{Bmatrix} identifier\text{-}1 \\ literal\text{-}1 \end{Bmatrix}$ INTO identifier-2 [ROUNDED]

[ON SIZE ERROR imperative-statement]

FORMAT 2

DIVIDE $\begin{Bmatrix} identifier\text{-}1 \\ literal\text{-}1 \end{Bmatrix}$ $\begin{Bmatrix} \underline{INTO} \\ \underline{BY} \end{Bmatrix}$ $\begin{Bmatrix} identifier\text{-}2 \\ literal\text{-}2 \end{Bmatrix}$ GIVING identifier-3 [ROUNDED]

[REMAINDER identifier-4] [ON SIZE ERROR imperative-statement]

ENTER Statement

ENTER language-name [routine-name].

ENTRY Statement

ENTRY literal-1 [USING identifier-1 [identifier-2] ...]

EXAMINE Statement

FORMAT 1

EXAMINE identifier TALLYING $\begin{Bmatrix} \underline{UNTIL} \ \underline{FIRST} \\ \underline{ALL} \\ \underline{LEADING} \end{Bmatrix}$ literal-1

[REPLACING BY literal-2]

FORMAT 2

EXAMINE identifier REPLACING $\begin{Bmatrix} \underline{ALL} \\ \underline{LEADING} \\ \underline{FIRST} \\ \underline{UNTIL} \ \underline{FIRST} \end{Bmatrix}$ literal-1 BY literal-2

EXIT Statement

paragraph-name. EXIT [PROGRAM].

GOBACK Statement

GOBACK.

GO TO Statement

FORMAT 1

GO TO procedure-name-1

FORMAT 2

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

FORMAT 3

GO TO.

IF Statement

IF condition THEN $\begin{Bmatrix} \underline{NEXT} \ \underline{SENTENCE} \\ statement\text{-}1 \end{Bmatrix}$ $\begin{Bmatrix} \underline{OTHERWISE} \\ \underline{ELSE} \end{Bmatrix}$ $\begin{Bmatrix} \underline{NEXT} \ \underline{SENTENCE} \\ statement\text{-}2 \end{Bmatrix}$

MOVE Statement

FORMAT 1

MOVE $\begin{Bmatrix} identifier\text{-}1 \\ literal \end{Bmatrix}$ TO identifier-2 [identifier-3] ...

FORMAT 2

MOVE $\begin{Bmatrix} \underline{CORRESPONDING} \\ \underline{CORR} \end{Bmatrix}$ identifier-1 TO identifier-2

6

IBM AMERICAN NATIONAL STANDARD COBOL RESERVED WORDS

No word in the following list should appear as a programmer defined name. The keys that appear before some of the words, and their meanings, are:

(xa) before a word means that the word is a System/360 extension to American National Standard COBOL.

(xac) before a word means that the word is a System/360 extension to both American National Standard COBOL and CODASYL COBOL.

(ca) before a word means that the word is a CODASYL COBOL reserved word not incorporated in American National Standard COBOL or in IBM American National Standard COBOL.

(sp) before a word means that the word is an IBM function-name established in support of the SPECIAL-NAMES function.

(spn) before a word means that the word is used by an IBM System/360 American National Standard COBOL compiler, but not this compiler.

(asn) before a word means that the word is defined by American National Standard COBOL, but is not used by this compiler.

	ACCEPT	(xac)	COM-REG
	ACCESS		COMMA
	ACTUAL		COMP
	ADD	(xa)	COMP-1
(asn)	ADDRESS	(xa)	COMP-2
	ADVANCING	(xa)	COMP-3
	AFTER	(xa)	COMP-4
	ALL		COMPUTATIONAL
	ALPHABETIC	(xa)	COMPUTATIONAL-1
(ca)	ALPHANUMERIC	(xa)	COMPUTATIONAL-2
(ca)	ALPHANUMERIC-EDITED	(xa)	COMPUTATIONAL-3
	ALTER	(xa)	COMPUTATIONAL-4
	ALTERNATE		COMPUTE
	AND		CONFIGURATION
(xa)	APPLY	(sp)	CONSOLE
	ARE		CONTAINS
	AREA		CONTROL
	AREAS		CONTROLS
	ASCENDING		COPY
	ASSIGN	(xac)	CORE-INDEX
	AT		CORR
	AUTHOR		CORRESPONDING
		(ca)	COUNT
(xac)	BASIS	(sp)	CSP
	BEFORE		CURRENCY
	BEGINNING	(xac)	CURRENT-DATE
	BLANK	(xac)	CYL-INDEX
	BLOCK	(xac)	CYL-OVERFLOW
	BY	(sp)	C01
		(sp)	C02
(xa)	CALL	(sp)	C03
(ca)	CANCEL	(sp)	C04
(spn)	CBL	(sp)	C05
(ca)	CD	(sp)	C06
	CF	(sp)	C07
	CH	(sp)	C08
(xac)	CHANGED	(sp)	C09
(xa)	CHARACTER	(sp)	C10
	CHARACTERS	(sp)	C11
(asn)	CLOCK-UNITS	(sp)	C12
	CLOSE		
(asn)	COBOL		DATA
	CODE	(ca)	DATE
	COLUMN		DATE-COMPILED

15

DEBUGGING LANGUAGE — BASIC FORMATS

Procedure Division Debugging Formats

EXHIBIT Statement

$$\underline{\text{EXHIBIT}} \left\{ \begin{array}{l} \underline{\text{NAMED}} \\ \underline{\text{CHANGED}} \ \underline{\text{NAMED}} \\ \underline{\text{CHANGED}} \end{array} \right\} \left\{ \begin{array}{l} identifier\text{-}1 \\ nonnumeric\text{-}literal\text{-}1 \end{array} \right\} \left[\begin{array}{l} identifier\text{-}2 \\ nonnumeric\text{-}literal\text{-}2 \end{array} \right] \cdots$$

ON (Count-conditional) Statement

FORMAT 1

$\underline{\text{ON}}$ integer-1 [$\underline{\text{AND}}$ $\underline{\text{EVERY}}$ integer-2] [$\underline{\text{UNTIL}}$ integer-3]

$$\left\{ \begin{array}{l} imperative\text{-}statement \ldots \\ \underline{\text{NEXT}} \ \underline{\text{SENTENCE}} \end{array} \right\} \left\{ \begin{array}{l} \underline{\text{ELSE}} \\ \underline{\text{OTHERWISE}} \end{array} \right\} \left\{ \begin{array}{l} statement \ldots \\ \underline{\text{NEXT}} \ \underline{\text{SENTENCE}} \end{array} \right\}$$

FORMAT 2 (Version 3)

$$\underline{\text{ON}} \left\{ \begin{array}{l} integer\text{-}1 \\ identifier\text{-}1 \end{array} \right\} [\underline{\text{AND}} \ \underline{\text{EVERY}} \left\{ \begin{array}{l} integer\text{-}2 \\ identifier\text{-}2 \end{array} \right\}]$$

$$[\underline{\text{UNTIL}} \left\{ \begin{array}{l} integer\text{-}3 \\ identifier\text{-}3 \end{array} \right\}] \left\{ \begin{array}{l} imperative\text{-}statement \\ \underline{\text{NEXT}} \ \underline{\text{SENTENCE}} \end{array} \right\}$$

$$\left\{ \begin{array}{l} \underline{\text{ELSE}} \\ \underline{\text{OTHERWISE}} \end{array} \right\} \left\{ \begin{array}{l} statement \ldots \\ \underline{\text{NEXT}} \ \underline{\text{SENTENCE}} \end{array} \right\}$$

READY/RESET TRACE Statement

$$\left\{ \begin{array}{l} \underline{\text{READY}} \\ \underline{\text{RESET}} \end{array} \right\} \ \underline{\text{TRACE}}$$

Compile-Time Debugging Packet

DEBUG Card

$\underline{\text{DEBUG}}$ location

FORMAT CONTROL — BASIC FORMATS

EJECT Statement

1	Area B
	$\underline{\text{EJECT}}$

SKIP1, SKIP2, SKIP3 Statements

1	Area B
	$\left\{ \begin{array}{l} \underline{\text{SKIP1}} \\ \underline{\text{SKIP2}} \\ \underline{\text{SKIP3}} \end{array} \right\}$

STERLING CURRENCY — BASIC FORMATS

Data Division Sterling Formats

Nonreport PICTURE Clause

$\left\{ \begin{array}{l} \underline{\text{PICTURE}} \\ \underline{\text{PIC}} \end{array} \right\}$ IS 9[(n)] D [8] 8D $\left\{ \begin{array}{l} 6[6] \\ 7[7] \end{array} \right\}$ [[V] 9 [(n)]] [$\underline{\text{USAGE}}$ IS] $\underline{\text{DISPLAY-ST}}$

Report PICTURE Clause

$\left\{ \begin{array}{l} \underline{\text{PICTURE}} \\ \underline{\text{PIC}} \end{array} \right\}$ IS [pound-report-string] [pound-separator-string] delimiter
shilling-report-string [shilling-separator-string] delimiter
pence-report-string [pence-separator-string] [sign-string]
[$\underline{\text{USAGE}}$ IS] $\underline{\text{DISPLAY-ST}}$

14

MULTIPLY Statement

FORMAT 1

$\underline{\text{MULTIPLY}} \left\{ \begin{array}{l} identifier\text{-}1 \\ literal\text{-}1 \end{array} \right\} \ \underline{\text{BY}}$ identifier-2 [$\underline{\text{ROUNDED}}$]

[ON $\underline{\text{SIZE}}$ $\underline{\text{ERROR}}$ imperative-statement]

FORMAT 2

$\underline{\text{MULTIPLY}} \left\{ \begin{array}{l} identifier\text{-}1 \\ literal\text{-}1 \end{array} \right\} \ \underline{\text{BY}} \left\{ \begin{array}{l} identifier\text{-}2 \\ literal\text{-}2 \end{array} \right\} \ \underline{\text{GIVING}}$ identifier-3

[$\underline{\text{ROUNDED}}$] [ON $\underline{\text{SIZE}}$ $\underline{\text{ERROR}}$ imperative-statement]

NOTE Statement

$\underline{\text{NOTE}}$ character string

OPEN Statement

$\underline{\text{OPEN}}$ [$\underline{\text{INPUT}}$ {file-name $\left[\begin{array}{l} \underline{\text{REVERSED}} \\ \text{WITH } \underline{\text{NO}} \ \underline{\text{REWIND}} \end{array} \right]$ } ...]

[$\underline{\text{OUTPUT}}$ {file-name [WITH $\underline{\text{NO}}$ $\underline{\text{REWIND}}$]} ...]

[$\underline{\text{I-O}}$ {file-name} ...]

PERFORM Statement

FORMAT 1

$\underline{\text{PERFORM}}$ procedure-name-1 [$\underline{\text{THRU}}$ procedure-name-2]

FORMAT 2

$\underline{\text{PERFORM}}$ procedure-name-1 [$\underline{\text{THRU}}$ procedure-name-2] $\left\{ \begin{array}{l} identifier\text{-}1 \\ integer\text{-}1 \end{array} \right\}$ $\underline{\text{TIMES}}$

FORMAT 3

$\underline{\text{PERFORM}}$ procedure-name-1 [$\underline{\text{THRU}}$ procedure-name-2] $\underline{\text{UNTIL}}$ condition-1

FORMAT 4

$\underline{\text{PERFORM}}$ procedure-name-1 [$\underline{\text{THRU}}$ procedure-name-2]

$\underline{\text{VARYING}} \left\{ \begin{array}{l} index\text{-}name\text{-}1 \\ identifier\text{-}1 \end{array} \right\} \ \underline{\text{FROM}} \left\{ \begin{array}{l} index\text{-}name\text{-}2 \\ literal\text{-}2 \\ identifier\text{-}2 \end{array} \right\} \ \underline{\text{BY}}$

$\left\{ \begin{array}{l} literal\text{-}3 \\ identifier\text{-}3 \end{array} \right\} \ \underline{\text{UNTIL}}$ condition-1

$[\underline{\text{AFTER}} \left\{ \begin{array}{l} index\text{-}name\text{-}4 \\ identifier\text{-}4 \end{array} \right\} \ \underline{\text{FROM}} \left\{ \begin{array}{l} index\text{-}name\text{-}5 \\ literal\text{-}5 \\ identifier\text{-}5 \end{array} \right\} \ \underline{\text{BY}}$

$\left\{ \begin{array}{l} literal\text{-}6 \\ identifier\text{-}6 \end{array} \right\} \ \underline{\text{UNTIL}}$ condition-2

$[\underline{\text{AFTER}} \left\{ \begin{array}{l} index\text{-}name\text{-}7 \\ identifier\text{-}7 \end{array} \right\} \ \underline{\text{FROM}} \left\{ \begin{array}{l} index\text{-}name\text{-}8 \\ literal\text{-}8 \\ identifier\text{-}8 \end{array} \right\} \ \underline{\text{BY}}$

$\left\{ \begin{array}{l} literal\text{-}9 \\ identifier\text{-}9 \end{array} \right\} \ \underline{\text{UNTIL}}$ condition-3]]

READ Statement

$\underline{\text{READ}}$ file-name RECORD [$\underline{\text{INTO}}$ identifier] $\left\{ \begin{array}{l} \text{AT } \underline{\text{END}} \\ \underline{\text{INVALID}} \ \text{KEY} \end{array} \right\}$ imperative-statement

REWRITE Statement

$\underline{\text{REWRITE}}$ record-name [$\underline{\text{FROM}}$ identifier] $\underline{\text{INVALID}}$ KEY imperative-statement

SEEK Statement

$\underline{\text{SEEK}}$ file-name RECORD

START Statement

START *file-name* INVALID KEY *imperative-statement*

STOP Statement

STOP $\begin{Bmatrix} \text{RUN} \\ \textit{literal} \end{Bmatrix}$

SUBTRACT Statement

FORMAT 1

SUBTRACT $\begin{Bmatrix} \textit{identifier-1} \\ \textit{literal-1} \end{Bmatrix} \begin{bmatrix} \textit{identifier-2} \\ \textit{literal-2} \end{bmatrix}$ ... FROM *identifier-m* [ROUNDED]

[*identifier-n* [ROUNDED]] ... [ON SIZE ERROR *imperative-statement*]

FORMAT 2

SUBTRACT $\begin{Bmatrix} \textit{identifier-1} \\ \textit{literal-1} \end{Bmatrix} \begin{bmatrix} \textit{identifier-2} \\ \textit{literal-2} \end{bmatrix}$ ... FROM $\begin{Bmatrix} \textit{identifier-m} \\ \textit{literal-m} \end{Bmatrix}$ GIVING

identifier-n [ROUNDED] [ON SIZE ERROR *imperative-statement*]

FORMAT 3

SUBTRACT $\begin{Bmatrix} \text{CORRESPONDING} \\ \text{CORR} \end{Bmatrix}$ *identifier-1* FROM *identifier-2* [ROUNDED]

[ON SIZE ERROR *imperative-statement*]

TRANSFORM Statement

TRANSFORM *identifier-3* CHARACTERS FROM $\begin{Bmatrix} \textit{figurative-constant-1} \\ \textit{nonnumeric-literal-1} \\ \textit{identifier-1} \end{Bmatrix}$

TO $\begin{Bmatrix} \textit{figurative-constant-2} \\ \textit{nonnumeric-literal-2} \\ \textit{identifier-2} \end{Bmatrix}$

USE Sentence

FORMAT 1

Option 1:

USE $\begin{Bmatrix} \text{BEFORE} \\ \text{AFTER} \end{Bmatrix}$ STANDARD [BEGINNING] $\begin{bmatrix} \text{REEL} \\ \text{FILE} \\ \text{UNIT} \end{bmatrix}$

LABEL PROCEDURE ON $\begin{Bmatrix} \{\textit{file-name}\} \text{ ...} \\ \text{OUTPUT} \\ \text{INPUT} \\ \text{I-O} \end{Bmatrix}$.

Option 2:

USE $\begin{Bmatrix} \text{BEFORE} \\ \text{AFTER} \end{Bmatrix}$ STANDARD [ENDING] $\begin{bmatrix} \text{REEL} \\ \text{FILE} \\ \text{UNIT} \end{bmatrix}$.

LABEL PROCEDURE ON $\begin{Bmatrix} \{\textit{file-name}\} \text{ ...} \\ \text{OUTPUT} \\ \text{INPUT} \\ \text{I-O} \end{Bmatrix}$.

FORMAT 2

USE AFTER STANDARD ERROR PROCEDURE

ON $\begin{Bmatrix} \{\textit{file-name-1}\} \\ \textit{file-name-2} \text{ GIVING } \textit{data-name-1} [\textit{data-name-2}] \\ \text{INPUT} \\ \text{OUTPUT} \\ \text{I-O} \end{Bmatrix}$.

NOTE: Format 3 of the USE Sentence is included in Formats for the REPORT WRITER feature.

8

SET Statement

FORMAT 1

SET $\begin{Bmatrix} \textit{index-name-1} & [\textit{index-name-2}] \text{ ...} \\ \textit{identifier-1} & [\textit{identifier-2} \quad] \text{ ...} \end{Bmatrix}$ TO $\begin{Bmatrix} \textit{index-name-3} \\ \textit{identifier-3} \\ \textit{literal-1} \end{Bmatrix}$

FORMAT 2

SET *index-name-4* [*index-name-5*] ... $\begin{Bmatrix} \text{UP BY} \\ \text{DOWN BY} \end{Bmatrix}$ $\begin{Bmatrix} \textit{identifier-4} \\ \textit{literal-2} \end{Bmatrix}$

SEGMENTATION — BASIC FORMATS

Environment Division Segmentation Formats

SEGMENT-LIMIT Clause
Object-Computer Paragraph

SEGMENT-LIMIT IS *priority-number*

Procedure Division Segmentation Formats

Priority Numbers

section-name SECTION [*priority-number*].

SOURCE PROGRAM LIBRARY FACILITY — BASIC FORMATS

COPY Statement

COPY *library-name* [SUPPRESS]

[REPLACING *word-1* BY $\begin{Bmatrix} \textit{word-2} \\ \textit{literal-1} \\ \textit{identifier-1} \end{Bmatrix}$ [*word-3* BY $\begin{Bmatrix} \textit{word-4} \\ \textit{literal-2} \\ \textit{identifier-2} \end{Bmatrix}$] ...] .

Extended Source Program Library Facility

BASIS Card

BASIS *library-name*

INSERT Card

INSERT *sequence-number-field*

DELETE Card

DELETE *sequence-number-field*

13

Procedure Division Report Writer Formats

GENERATE Statement

GENERATE *identifier*

INITIATE Statement

INITIATE *report-name-1* [*report-name-2*] . . .

TERMINATE Statement

TERMINATE *report-name-1* [*report-name-2*] . . .

USE Sentence

USE BEFORE REPORTING *identifier-1*.

TABLE HANDLING — BASIC FORMATS

Data Division Table Handling Formats

OCCURS Clause

FORMAT 1

OCCURS *integer-2* TIMES

[{ASCENDING / DESCENDING} KEY IS *data-name-2* [*data-name-3*] . . .] . . .

[INDEXED BY *index-name-1* [*index-name-2*] . . .]

FORMAT 2

OCCURS *integer-1* TO *integer-2* TIMES [DEPENDING ON *data-name-1*]

[{ASCENDING / DESCENDING} KEY IS *data-name-2* [*data-name-3*] . . .] . . .

[INDEXED BY *index-name-1* [*index-name-2*] . . .]

FORMAT 3

OCCURS *integer-2* TIMES [DEPENDING ON *data-name-1*]

[{ASCENDING / DESCENDING} KEY IS *data-name-2* [*data-name-3*] . . .] . . .

[INDEXED BY *index-name-1* [*index-name-2*] . . .]

USAGE Clause

[USAGE IS] INDEX

Procedure Division Table Handling Formats

SEARCH Statement

FORMAT 1

SEARCH *identifier-1* [VARYING {*index-name-1* / *identifier-2*}]

[AT END *imperative-statement-1*]

WHEN *condition-1* {*imperative-statement-2* / NEXT SENTENCE}

[WHEN *condition-2* {*imperative-statement-3* / NEXT SENTENCE}] . . .

FORMAT 2

SEARCH ALL *identifier-1* [AT END *imperative-statement-1*]

WHEN *condition-1* {*imperative-statement-2* / NEXT SENTENCE}

12

WRITE Statement

FORMAT 1

WRITE *record-name* [FROM *identifier-1*] [{BEFORE / AFTER} ADVANCING

{ *identifier-2* LINES / *integer* LINES / *mnemonic-name* }] [AT {END-OF-PAGE / EOP} *imperative-statement*]

FORMAT 2

WRITE *record-name* [FROM *identifier-1*] AFTER POSITIONING {*identifier-2* / *integer*} LINES

[AT {END-OF-PAGE / EOP} *imperative-statement*]

FORMAT 3

WRITE *record-name* [FROM *identifier-1*] INVALID KEY *imperative-statement*

SORT — BASIC FORMATS

Environment Division Sort Formats

FILE-CONTROL PARAGRAPH — SELECT SENTENCE

SELECT Sentence (for GIVING option only)

SELECT *file-name*

ASSIGN TO [*integer-1*] *system-name-1* [*system-name-2*] . . .

OR *system-name-3* [FOR MULTIPLE {REEL / UNIT}]

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

SELECT Sentence (for Sort Work Files)

SELECT *sort-file-name*

ASSIGN TO [*integer*] *system-name-1* [*system-name-2*] . . . .

I-O CONTROL PARAGRAPH

RERUN Clause

RERUN ON *system-name*

SAME RECORD/SORT AREA Clause

SAME {RECORD / SORT} AREA FOR *file-name 1* {*file-name-2*} . . .

Data Division Sort Formats

SORT-FILE DESCRIPTION

SD *sort-file-name*

RECORDING MODE IS *mode*

DATA {RECORD IS / RECORDS ARE} *data-name-1* [*data-name-2*] . . .

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

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

9

Procedure Division Sort Formats

RELEASE Statement

RELEASE sort-record-name [FROM identifier]

RETURN Statement

RETURN sort-file-name RECORD [INTO identifier]

        AT END imperative-statement

SORT Statement

SORT file-name-1 ON $\left\{ \begin{array}{l} \text{DESCENDING} \\ \text{ASCENDING} \end{array} \right\}$ KEY {data-name-1} . . .

      [ON $\left\{ \begin{array}{l} \text{DESCENDING} \\ \text{ASCENDING} \end{array} \right\}$ KEY {data-name-2} . . .] . . .

      $\left\{ \begin{array}{l} \text{INPUT PROCEDURE IS section-name-1 [THRU section-name-2]} \\ \text{USING file-name-2} \end{array} \right\}$

      $\left\{ \begin{array}{l} \text{OUTPUT PROCEDURE IS section-name-3 [THRU section-name-4]} \\ \text{GIVING file-name-3} \end{array} \right\}$

REPORT WRITER — BASIC FORMATS

Data Division — Report Writer Formats

NOTE: Formats which appear as Basic Formats within the general description of the Data Division are illustrated there.

FILE SECTION — REPORT Clause

$\left\{ \begin{array}{l} \text{REPORT IS} \\ \text{REPORTS ARE} \end{array} \right\}$ report-name-1 [report-name-2] . . .

REPORT SECTION

REPORT SECTION.

RD report-name

   WITH CODE mnemonic-name

  $\left\{ \begin{array}{l} \text{CONTROL IS} \\ \text{CONTROLS ARE} \end{array} \right\}$ $\left\{ \begin{array}{l} \text{FINAL} \\ \text{identifier-1 [identifier-2] . . .} \\ \text{FINAL identifier-1 [identifier-2] . . .} \end{array} \right\}$

  PAGE $\left[\begin{array}{l} \text{LIMIT IS} \\ \text{LIMITS ARE} \end{array} \right]$ integer-1 $\left\{ \begin{array}{l} \text{LINE} \\ \text{LINES} \end{array} \right\}$

  [HEADING integer-2]

  [FIRST DETAIL integer-3]

  [LAST DETAIL integer-4]

  [FOOTING integer-5] .

10

REPORT GROUP DESCRIPTION ENTRY

FORMAT 1

01 [data-name-1]

  LINE NUMBER IS $\left\{ \begin{array}{l} \text{integer-1} \\ \text{PLUS integer-2} \\ \text{NEXT PAGE} \end{array} \right\}$

  NEXT GROUP IS $\left\{ \begin{array}{l} \text{integer-1} \\ \text{PLUS integer-2} \\ \text{NEXT PAGE} \end{array} \right\}$

  TYPE IS $\left\{ \begin{array}{l} \left\{ \begin{array}{l} \text{REPORT HEADING} \\ \text{RH} \end{array} \right\} \\ \left\{ \begin{array}{l} \text{PAGE HEADING} \\ \text{PH} \end{array} \right\} \\ \left\{ \begin{array}{l} \text{CONTROL HEADING} \\ \text{CH} \end{array} \right\} \left\{ \begin{array}{l} \text{identifier-n} \\ \text{FINAL} \end{array} \right\} \\ \left\{ \begin{array}{l} \text{DETAIL} \\ \text{DE} \end{array} \right\} \\ \left\{ \begin{array}{l} \text{CONTROL FOOTING} \\ \text{CF} \end{array} \right\} \left\{ \begin{array}{l} \text{identifier-n} \\ \text{FINAL} \end{array} \right\} \\ \left\{ \begin{array}{l} \text{PAGE FOOTING} \\ \text{PF} \end{array} \right\} \\ \left\{ \begin{array}{l} \text{REPORT FOOTING} \\ \text{RF} \end{array} \right\} \end{array} \right\}$

  USAGE Clause.

FORMAT 2

nn [data-name-1]

  LINE Clause — See Format 1

  USAGE Clause.

FORMAT 3

nn [data-name-1]

  BLANK WHEN ZERO Clause

  COLUMN NUMBER IS integer-1

  GROUP INDICATE

  JUSTIFIED Clause

  LINE Clause — See Format 1

  PICTURE Clause

  RESET ON $\left\{ \begin{array}{l} \text{identifier-1} \\ \text{FINAL} \end{array} \right\}$

  SOURCE IS $\left\{ \begin{array}{l} \text{TALLY} \\ \text{identifier-2} \end{array} \right\}$

  SUM $\left\{ \begin{array}{l} \text{TALLY} \\ \text{identifier-3} \end{array} \right\}$ $\left\{ \begin{array}{l} \text{TALLY} \\ \text{identifier-4} \end{array} \right\}$ . . . [UPON data-name]

  VALUE IS literal-1

  USAGE Clause.

FORMAT 4

01 data-name-1

  BLANK WHEN ZERO Clause

  COLUMN Clause — See Format 2

  GROUP Clause — See Format 2

  JUSTIFIED Clause

  LINE Clause — See Format 1

  NEXT GROUP Clause — See Format 1

  PICTURE Clause

  RESET Clause — See Format 2

  $\left\{ \begin{array}{l} \text{SOURCE Clause} \\ \text{SUM Clause} \\ \text{VALUE Clause} \end{array} \right\}$ See Format 2

  TYPE Clause — See Format 1

  USAGE Clause.

11

This appendix summarizes the statements and clauses that may be
specified for each file-processing technique. In addition, each
file-name must be specified in a SELECT clause in the Environment
Division and must be defined by an FD entry in the File Section of the
Data Division.

INDEXED FILES (mass storage devices only) — Required and Optional Entries

	Required Entries						Optional Entries					
ACCESS	System-name	KEY	LABEL RECORDS	OPEN	CLOSE	Access Verbs	APPLY	RESERVE	Other ENVIRONMENT DIVISION Clauses	RECORDING MODE	BLOCK CONTAINS	USE
[SEQUENTIAL]	SYSnnn-DA-xxxx-I [-name]	RECORD	STANDARD	INPUT	[LOCK]	READ [INTO] AT END	{MASTER-INDEX}{CYL-INDEX}	NO	SAME [RECORD] AREA RERUN	F	n RECORDS	ERROR
		[NOMINAL]		OUTPUT	[LOCK]	[START INVALID KEY] / WRITE[1] [FROM] INVALID KEY	CYL-OVERFLOW {MASTER-INDEX}{CYL-INDEX} WRITE-VERIFY					
		[NOMINAL]		I-O	[LOCK]	READ [INTO] INVALID KEY REWRITE[2] [FROM] INVALID KEY [START INVALID KEY]						
RANDOM	SYSnnn-DA-xxxx-I [-name]	RECORD NOMINAL	STANDARD	INPUT	[LOCK]	READ [INTO] INVALID KEY	CORE-INDEX {MASTER-INDEX}{CYL-INDEX}	NO	SAME [RECORD] AREA RERUN	F	n RECORDS	ERROR
				I-O	[LOCK]	READ [INTO] INVALID KEY WRITE[3] [FROM] INVALID KEY REWRITE[2] [FROM] INVALID KEY	CORE-INDEX CYL-OVERFLOW {MASTER-INDEX}{CYL-INDEX} WRITE-VERIFY					

[1]Create [2]Update [3]Add

STANDARD SEQUENTIAL FILES – Required and Optional Entries

Device Type		Required Entries				Optional Entries						
	System-name	LABEL RECORDS	OPEN	CLOSE	Access Verbs	APPLY	RESERVE	ACCESS	Other ENVIRONMENT DIVISION Clauses	BLOCK CONTAINS[3]	RECORDING MODE	USE
Reader	SYSnnn-UR-xxxx-S [-name]	OMITTED	INPUT	[LOCK]	READ [INTO] AT END	—	$\begin{Bmatrix} integer \\ NO \end{Bmatrix}$	SEQUENTIAL	SAME [RECORD] AREA RERUN	—	F	ERROR[4]
Punch	SYSnnn-UR-xxxx-S [-name]	OMITTED	OUTPUT	[LOCK]	WRITE[1] [FROM] [$\begin{Bmatrix} BEFORE \\ AFTER \end{Bmatrix}$ ADVANCING] [AFTER POSITIONING]	—	$\begin{Bmatrix} integer \\ NO \end{Bmatrix}$	SEQUENTIAL	SAME [RECORD] AREA RERUN	—	$\begin{Bmatrix} F \\ U \\ V \end{Bmatrix}$	ERROR[4]
Printer	SYSnnn-UR-xxxx-S [-name]	OMITTED	OUTPUT	[LOCK]	WRITE[1] [FROM] [$\begin{Bmatrix} BEFORE \\ AFTER \end{Bmatrix}$ ADVANCING] [AFTER POSITIONING] [END-OF-PAGE]	—	$\begin{Bmatrix} integer \\ NO \end{Bmatrix}$ NO	SEQUENTIAL	SAME [RECORD] AREA RERUN	—	$\begin{Bmatrix} F \\ U \\ V \end{Bmatrix}$	ERROR[4] REPORTING
Tape	SYSnnn-UT-xxxx-S [-name]	$\begin{Bmatrix} STANDARD \\ OMITTED \\ data\text{-}name \end{Bmatrix}$	INPUT [REVERSED] [NO REWIND]	[REEL] [LOCK NO REWIND]	READ [INTO] AT END	—	$\begin{Bmatrix} integer \\ NO \end{Bmatrix}$	SEQUENTIAL	SAME [RECORD] AREA RERUN MULTIPLE FILE TAPE	[n TO] m	$\begin{Bmatrix} F \\ U \\ V \\ S \end{Bmatrix}$	LABEL ERROR
			OUTPUT [NO REWIND]	[REEL] [LOCK NO REWIND]	WRITE[1] [FROM] [$\begin{Bmatrix} BEFORE \\ AFTER \end{Bmatrix}$ ADVANCING] [AFTER POSITIONING]	WRITE-ONLY						LABEL ERROR REPORTING
Mass Storage	SYSnnn-$\begin{Bmatrix} UT \\ DA \end{Bmatrix}$-xxxx-S [-name]	$\begin{Bmatrix} STANDARD \\ data\text{-}name \end{Bmatrix}$	INPUT	[UNIT] [LOCK]	READ [INTO] AT END	—	$\begin{Bmatrix} integer \\ NO \end{Bmatrix}$	SEQUENTIAL	SAME [RECORD] AREA RERUN	[n TO] m	$\begin{Bmatrix} F \\ U \\ V \\ S \end{Bmatrix}$	AFTER LABEL ERROR
			OUTPUT	[UNIT] [LOCK]	WRITE[1] [FROM] [$\begin{Bmatrix} BEFORE \\ AFTER \end{Bmatrix}$ ADVANCING] [AFTER POSITIONING] WRITE[1] [FROM] INVALID KEY	WRITE-ONLY WRITE-VERIFY						AFTER LABEL ERROR REPORTING
			I-O	[UNIT] [LOCK]	READ [INTO] AT END WRITE[2] [FROM] INVALID KEY	WRITE-VERIFY						AFTER LABEL ERROR

[1]Create   [2]Update   [3]Not for U mode   [4] STXIT Option must be specified on the CBL control card

Appendix D: Summary of File-Processing Techniques 333

DIRECT FILES (mass storage devices only) – Required and Optional Entries

	Required Entries						Optional Entries			
ACCESS	KEY	System-name	LABEL RECORDS	OPEN	CLOSE	Access Verbs	APPLY	Other ENVIRONMENT DIVISION Clauses	RECORDING MODE	USE
[SEQUENTIAL]	[ACTUAL]	SYSnnn-DA-xxxx-$\{^A_D\}$ [-name]	$\{^{STANDARD}_{data\text{-}name}\}$	INPUT	[UNIT] [LOCK]	READ [INTO] AT END		SAME [RECORD] AREA RERUN	$\{^F_{U}_{S}\}$	AFTER LABEL ERROR
RANDOM	ACTUAL	SYSnnn-DA-xxxx-$\{^A_D\}$ [-name]	$\{^{STANDARD}_{data\text{-}name}\}$	INPUT	[LOCK]	SEEK READ [INTO] INVALID KEY	EXTENDED-SEARCH	SAME [RECORD] AREA RERUN	$\{^F_{U}_{S}\}$	AFTER LABEL ERROR
				OUTPUT	[LOCK]	SEEK WRITE[1] [FROM] INVALID KEY	WRITE-VERIFY			
				I-O	[LOCK]	SEEK READ [INTO] INVALID KEY WRITE[2] [FROM] INVALID KEY	EXTENDED-SEARCH WRITE-VERIFY			
RANDOM	ACTUAL	SYSnnn-DA-xxxx-$\{^U_W\}$ [-name]	$\{^{STANDARD}_{data\text{-}name}\}$	INPUT	[LOCK]	SEEK READ [INTO] INVALID KEY	EXTENDED-SEARCH	SAME [RECORD] AREA RERUN	$\{^F_{U}_{S}\}$	AFTER LABEL ERROR
				OUTPUT	[LOCK]	SEEK WRITE[2] [FROM] INVALID KEY	WRITE-VERIFY			
RANDOM	ACTUAL	SYSnnn-DA-xxxx-$\{^U_W\}$ [-name]	$\{^{STANDARD}_{data\text{-}name}\}$	I-O	[LOCK]	SEEK READ [INTO] INVALID KEY	EXTENDED-SEARCH	SAME [RECORD] AREA RERUN	$\{^F_{U}_{S}\}$	AFTER LABEL ERROR
						WRITE[3] [FROM] INVALID KEY	WRITE-VERIFY			
						REWRITE[4] [FROM] INVALID KEY	EXTENDED-SEARCH WRITE-VERIFY			

[1]Create   [2]Update and add   [3]Add   [4]Update

This compiler supports the American National Standard Code for Information Interchange (ASCII). Thus the programmer can create and process tape files recorded in accordance with the following standards:

- ASCII Standard Code X3.4-1967

- American National Standard X3.27-1969, Magnetic Tape Labels for Information Interchange

- American National Standard 9-track, 800 bpi, NRZI Magnetic Tape Standard X3.22-1967

ASCII encoded tape files, when read into the system, are automatically translated in the buffers into EBCDIC. Internal manipulation of data is performed exactly as if they were EBCDIC encoded files. For an output file, the system translates the EBCDIC characters into ASCII in the buffers before writing the file out on tape. Therefore there are special considerations concerning ASCII encoded files when they are processed in COBOL. The following paragraphs discuss these considerations.

I -- ENVIRONMENT DIVISION

Environment Division clauses affected by the specification of ASCII files are the ASSIGN clause and the RERUN clause.

ASSIGN Clause

When ASCII files are to be processed, the system-name in the ASSIGN clause has the following format:

    SYSnnn-UT-device-C[-offset][-name]

nnn is a three-digit number between 000 and 221. This number represents the symbolic unit to which the file is assigned.

UT for utility must be specified in the class field

device must specify a magnetic tape device (2400).

C in the organization field specifies that an ASCII encoded sequential file is to be processed, or that an ASCII collated sort is to be performed.

offset may be specified only for an ASCII file, and then only if a block prefix of length 01 through 99 exists. It is a 2-digit field, and may be specified as follows:

    01 through 99   for an input file
    04   for an output file (D-mode records only)

name is a one- to seven-character field specifying the external-name by which the file is known to the system. If specified, it is the name that appears in the file-name field of the VOL, DLBL, or TLBL job control statement. If this field is not specified, the symbolic

unit (SYSnnn) is used as the external-name. This field must be specified if more than one file is assigned to the same symbolic unit.

RERUN Clause

The system-name in a RERUN clause must not specify an ASCII encoded file.

ASCII encoded files containing checkpoint records cannot be processed.

II -- DATA DIVISION

In the Data Division there are special considerations for ASCII files, both in the File Section and in Data Description Entries.

FILE SECTION

In the File Section the BLOCK CONTAINS clause, the LABEL RECORDS clause and the RECORDING MODE clause are affected. There are also special considerations regarding the compiler default options for recording mode.

BLOCK CONTAINS Clause

For an ASCII file that contains a buffer offset field, the following considerations apply:

* If the BLOCK CONTAINS clause with the RECORDS option is specified, or if the BLOCK CONTAINS clause is omitted, the compiler compensates for the buffer offset field.

* If the BLOCK CONTAINS clause with the CHARACTERS option is specified, the programmer must include the buffer offset as part of the physical record.

LABEL RECORDS Clause

All three options of the clause (OMITTED/STANDARD/data-name) are allowed. However, if the programmer specifies the data-name option, he must make sure that data-name refers only to user standard labels. Nonstandard labels are not allowed for ASCII files.

RECORDING MODE Clause

For ASCII files, mode may be specified as F, U, or V. S mode may not be specified.

Compiler Calculation of Recording Mode

When the RECORDING MODE clause is not used to specify the mode of the records in an ASCII file, the COBOL compiler determines the mode by scanning each record description entry. The default option may be:

F   if all the records are defined as being the same size.

D   if the records are defined as variable in size, or if the RECORD CONTAINS clause specifies variable size records. Internally D mode is the equivalent of V mode for EBCDIC encoded files.

DATA DESCRIPTION ENTRIES

For ASCII files the Data Description Entries affected are the PICTURE clause, the SIGN clause, and the USAGE clause.

PICTURE Clause

For ASCII files all five categories of data are valid.

If a data item is numeric, however, and the item is signed, then the SIGN clause with the SEPARATE CHARACTER option must also be specified.

SIGN Clause

If a data item in an ASCII file is numeric and has a sign, then the SIGN clause with the SEPARATE CHARACTER option must be specified.

USAGE Clause

For data items in ASCII files, only the DISPLAY option of the USAGE clause is valid.

III -- PROCEDURE DIVISION

For ASCII files, there are special considerations in regard to Label Declaratives and relation conditions.

LABEL PROCEDURE Declarative

Since the user may not specify nonstandard labels for an ASCII encoded file, the BEFORE option of the LABEL PROCEDURE Declarative is not allowed.

Relation Conditions

If the ASCII character strings to be compared contain mixed alphabetic/numeric characters and/or special characters, then the TRANSFORM verb can be used before the comparison is made to ensure a valid comparison.

The following example illustrates a method of making the comparison (Figure 20 shows the necessary COBOL statements).

Suppose that the COBOL programmer specifies that one alphanumeric data item (ASCII-1) from ASCII-FILE is to be compared with another such data item (ASCII-2), and that the results of the comparison determine the path of program execution. Each data item may contain any valid COBOL character.

When ASCII-RECORD is read into the buffer, the system changes each ASCII character into its EBCDIC equivalent. Therefore, before a valid ASCII comparison can be made, the relative position of each character in the ASCII collating sequence must be reestablished.

In the Working-Storage Section, the VALUE of IDENT-EBCDIC is the ascending EBCDIC collating sequence (as shown in Figure 21); similarly, the VALUE of IDENT-ASCII is the ascending ASCII collating sequence. The contents of ASCII-1 are moved to DN-1, and the contents of ASCII-2 are moved to DN-2, and DN-1 and DN-2 are then used in the two TRANSFORM statements. (This avoids the necessity of a second pair of TRANSFORM statements to restore the original contents of ASCII-1 and ASCII-2.)

When the two TRANSFORM statements are executed, each EBCDIC character is exchanged for another EBCDIC character that occupies the original ASCII character's position in the ASCII collating sequence. Thus, when the comparison is made, it is valid for the ASCII collating sequence.

(Note that if ASCII-1 and ASCII-2 are restricted to mixed alphabetic and numeric characters, then the VALUE clauses in IDENT-EBCDIC and IDENT-ASCII need only contain alphabetic and numeric characters from the collating sequences. Note too that in the VALUE clause when quotation marks (") are used as delimiters, then the quotation mark itself cannot be one of the literals contained within the delimiter; similarly, if the apostrophe (') is used as the delimiter, then the apostrophe cannot be contained within the delimiter.)

```
                  .
                  .
                  .
|DATA DIVISION.
|FILE SECTION.
|FD   ASCII-FILE ...
|01   ASCII-RECORD.
|     05   ASCII-1 ...
|     05   ASCII-2 ...
|                  .
|                  .
|                  .
|WORKING-STORAGE SECTION.
|77   IDENT-ASCII   PICTURE X(51)   VALUE
|        " $'()*+,-./0123456789;<=>ABCDEFGHIJKLMNOPQRSTUVWXYZ".
|77   IDENT-EBCDIC   PICTURE X(51)   VALUE
|        " .<(+$*);-/,>'=ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789".
|77   DN-1 ...
|77   DN-2 ...
|                  .
|                  .
|                  .
|PROCEDURE DIVISION.
|                  .
|                  .
|                  .
|TEST-ASCII.
|     MOVE ASCII-1 TO DN-1.
|     MOVE ASCII-2 TO DN-2.
|     TRANSFORM DN-1 FROM IDENT-EBCDIC TO IDENT-ASCII.
|     TRANSFORM DN-2 FROM IDENT-EBCDIC TO IDENT-ASCII.
|     IF DN-1 IS GREATER THAN OR EQUAL TO DN-2
|        PERFORM PROCESS-1
|        ELSE PERFORM PROCESS-2.
|                  .
|                  .
|                  .
|PROCESS-1.
|                  .
|                  .
|                  .
|PROCESS-2.
|                  .
|                  .
|                  .
```

Figure 20.   Using the TRANSFORM Statement with ASCII Comparisons

EBCDIC Collating Sequence	ASCII Collating Sequence
1. (space)	1. (space)
2. . (period, decimal point)	2. " (quotation mark)
3. < (less than)	3. $ (currency symbol)
4. ((left parenthesis)	4. ' (apostrophe, single quotation mark)
5. + (plus symbol)	5. ((left parenthesis)
6. $ (currency symbol)	6.) (right parenthesis)
7. * (asterisk)	7. * (asterisk)
8.) (right parenthesis)	8. + (plus symbol)
9. ; (semicolon)	9. , (comma)
10. - (hyphen, minus symbol)	10. - (hyphen, minus symbol)
11. / (stroke, virgule, slash)	11. . (period, decimal point)
12. , (comma)	12. / (stroke, virgule, slash)
13. > (greater than)	
14. ' (apostrophe, single quotation mark)	13-22. 0 through 9
15. = (equal sign)	
16. " (quotation mark)	
	23. ; (semicolon)
	24. < (less than)
17-42. A through Z	25. = (equal sign)
	26. > (greater than)
43-52. 0 through 9	27-52. A through Z

Figure 21.   EBCDIC and ASCII Collating Sequences for COBOL
             Characters -- in ascending order

IV -- SORT FEATURE

For ASCII-collated sorts, there are special considerations in the
Environment Division and in the Data Division.

ENVIRONMENT DIVISION

For ASCII-collated sorts, there are special considerations for the
ASSIGN clause and for the RERUN clause.

ASSIGN Clause

The ASSIGN clause for an ASCII collated sort has the same format as
for an EBCDIC collated sort. However, the system-name must be in
the following format:

        SYSnnn-class-device-C-name

The following considerations apply:

SYSnnn must specify the fixed sort work units assigned to the
sort-file. The first work unit for every sort-file in the program
must be assigned to SYS001, the second to SYS002, etc.

class may be specified as UT or DA.

<u>device</u> may specify a utility or mass storage device.

<u>C</u> in the <u>organization</u> field specifies an ASCII collated sort.

<u>name</u> specifies the external-name by which the sort-file is known to the system. If the file has standard labels, the name field must be specified as SORTWK1 for SYS001, SORTWK2 for SYS002, etc.

<u>Note</u>: For an ASCII-collated sort, the buffer <u>offset</u> field is not permitted.

RERUN Clause

Checkpoint records for ASCII-collated sorts can be taken. However, the system-name specified in the RERUN clause must not specify an ASCII-encoded file.

DATA DIVISION

For ASCII-collated sorts, there are special considerations for the SIGN clause and for the USAGE clause.

SIGN Clause

If an S is specified in the PICTURE of a numeric item to be used as a sort key in an ASCII-collated sort, the SEPARATE option of the SIGN clause must be specified.

USAGE Clause

If an ASCII-collated sort is requested, the sort keys must be DISPLAY items, explicitly or implicitly.

Program Product Information -- Version 3

APPENDIX F: SYMBOLIC DEBUGGING FEATURE

A programmer using IBM Full American National Standard COBOL, Version 3, under the Disk Operating System, has several methods available to him for testing and debugging his programs. Use of the symbolic debugging feature is the easiest and most efficient method for testing and debugging and is described in detail in this appendix.

The symbolic debug option produces a symbolic formatted dump of the object program's data area when the program abnormally terminates. It also enables the programmer to request dynamic dumps of specific data-names at strategic points during program execution. If two or more COBOL programs are link edited together and one of them terminates abnormally, the program causing termination and any callers compiled with the symbolic debug option, up to and including the main program, will be given a formatted dump.

The abnormal termination dump consists of the following parts:

1. Abnormal termination message, including the number of the statement and of the verb being executed at the time of an abnormal termination.

2. Selected areas in the Task Global Table.

3. Formatted dump of the Data Division including:

    (a) for an SD, the card number, the sort-file-name, the type, and the sort record.

    (b) for an FD, the card number, the file-name, the type, SYSnnn, DTF status, the contents of the Pre-DTF and DTF in hexadecimal, and the fields of the record.

    (c) for an RD, the card number, the report-name, the type, the report line, and the contents of PAGE-COUNTER and LINE-COUNTER if present.

    (d) For an index-name, the name, the type, and the contents in decimal.

Note: For DTFDA when ACCESS IS RANDOM, the actual key is not provided in the Pre-DTF.

Operation of the symbolic debug option is dependent on object-time control cards placed in the input stream. These cards are discussed below.

Object-Time Control Cards

The operation of the symbolic debug option is determined by two types of control cards:

Program-control card -- required if abnormal termination and/or dynamic dumps are requested.

Line-control card -- required only if dynamic dumps are requested.

Program-Control Cards: A program-control card must be present at execution time for any program requesting symbolic debugging. A program-control card must contain the following information:

The 1-8 character program-name of the COBOL program compiled using symbolic debugging.

The logical unit and file-name assigned to the file produced at compile time on SYS005.

Additional optional parameters can also be specified:

An entry used to provide a trace of a program-name when several programs are link edited together. Each time the specified program is entered, its program name is displayed.

Two formats of the Data Division area in the abnormal termination dump are allowed:

1. Level-01 items are provided in hexadecimal. Items subordinate to level-01 items are printed in EBCDIC if possible. Level-77 items are printed both in hexadecimal and EBCDIC.

2. Level-77 items and items subordinate to level-01 items are provided in EBCDIC. If these items contain unprintable characters, hexadecimal notation is provided. This is the default option.

Line-Control Cards: A line-control card must contain the following information:

The card number associated with the point in the Procedure Division at which the dynamic dump is to be taken. The number specified is the compiler-generated card number.

Additional optional parameters can also be specified:

The position of the verb in the specified line number at which the dynamic dump is to be taken. When the verb position is not specified, the first verb in the line is assumed. Any verb position not exceeding 15 may be specified.

An equivalent to the COBOL statement "ON n AND EVERY m UNTIL k ...". This option limits the request dynamic dumps to specified times. For example "ON n" results in one dump, produced the nth time the line number is reached during execution. "ON n AND EVERY m" results in a dump the first time at the nth execution of the specified line number, and thereafter at every mth execution until end-of-job.

Two formats of the Data Division areas displayed in the dynamic dump are allowed:

1. Level-01 items are provided in hexadecimal. Items subordinate to level-01 items are provided in EBCDIC, if possible. Level-77 items are provided both in hexadecimal and EBCDIC.

2. Level-77 items subordinate to level-01 items are provided in EBCDIC. If these items contain unprintable characters, hexadecimal notation is provided. Note that if a group item is specified, neither the group nor the elementary items in the group are provided in hexadecimal. This is the default option.

Selected areas of the Data Division to be dumped. A single
data-name or a range of consecutive data-names can be
specified. (If the programmer wishes to see a subscripted
item, he specifies the name of the item without the subscript;
this results in a dump of every occurrence of the subscripted
item.)

A dump of everything that would be dumped in the event of an
abnormal termination can also be specified. This allows the
programmer to receive a formatted dump at normal end-of-job.
To do this, the programmer must specify the generated statement
number of the STOP RUN, GOBACK, or EXIT PROGRAM statement.

Sample Program -- TESTRUN

Figure 22 is an illustration of a program that utilizes the
symbolic debugging features. In the following description of the
program and its output, letters identifying the text correspond to
letters in the program listing.

(A) Because the SYMDMP option is requested on the CBL card, the
logical unit SYS005 must be assigned at compile time.

(B) The CBL card specifications indicate that an alphabetically
ordered cross-reference dictionary, a flow trace of 10
procedures, and the symbolic debug option are being requested.

(C) An alphabetically ordered cross-reference dictionary of
data-names and procedure-names is produced by the compiler as a
result of the SXREF specification on the CBL card.

(D) The file assigned at compile time to SYS005 to store SYMDMP
information is assigned to SYS009 at execution time.

(E) The SYMDMP control cards placed in the input stream at
execution time are printed along with any diagnostics.

(1) The first card is the program-control card where:

(a) TESTRUN is the PROGRAM-ID.

(b) 9 is the logical unit to which the SYMDMP file is
assigned.

(c) MT indicates that the SYMDMP file is on tape.

(d) (HEX) indicates the format of the abnormal termination
dump.

(2) The second card is a line-control card which requests a
(HEX) formatted dynamic dump of COUNT, NAME-FIELD,
NO-OF-DEPENDENTS, and RECORD-NO prior to the first and
every fourth execution of generated card number 71.

(3) The third card is also a line-control card which requests a
(HEX) formatted dynamic dump of WORK-RECORD and B prior to
the execution of generated card number 80.

(F) The type code combinations used to identify data-names in
abnormal termination and dynamic dumps are defined. Individual
codes are illustrated in Table 28.

(G) The dynamic dumps requested by the first line-control card.

(H) The dynamic dumps requested by the second line-control card.

(I) Program interrupt information is provided by the system when a program terminates abnormally.

(J) The statement number information indicates the number of the verb and of the statement being executed at the time of the abnormal termination. The name of the program containing the statement is also provided.

(K) A flow trace of the last 10 procedures executed is provided because FLOW=10 was specified on the CBL card.

(L) Selected areas of the Task Global Table are provided as part of the abnormal termination dump.

(M) For each file-name, the generated card number, the file type, SYSnnn, the DTF status, and the fields of the Pre-DTF and DTF in hexadecimal are provided.

(N) The fields of records associated with each FD are provided in the format requested on the program-control card.

(P) The contents of the fields of the Working-Storage Section are provided in the format requested on the program-control card.

(Q) The values associated with each of the possible subscripts are provided for data items described with an OCCURS clause.

(R) Asterisks appearing within the EBCDIC representation of the value of a given field indicate that the type and the actual content of the field conflict.

Note: When using the symbolic debugging option, level numbers appear "normalized" in the symbolic dump produced. For example, a group of data items described as:

```
    01   RECORDA.
         05 FIELD-A.
            10   FIELD-A1 PIC X.
            10   FIELD-A2 PIC X.
```

will appear as follows in symbolic debugging output:

```
    01 RECORDA...
    02 FIELD-A...
    03 FIELD-A1...
    03 FIELD-A2...
```

Debugging TESTRUN

1. Referring to the statement number information (J) provided by the symbolic debug option, it is learned that the abend occurred during the execution of the first verb on card 80.

2. Generated card number 80 contains the statement COMPUTE B = B + 1.

3. Verifying the contents of B at the time of the abnormal termination (R) it can be seen that the usage of B (numeric packed) conflicts with the value contained in the data area reserved for B (numeric display).

4. The abnormal termination occurred while trying to perform an addition on a display item.

More complex errors may require the use of dynamic dumps to isolate the problem area. Line-control cards are included in TESTRUN merely to illustrate how they are used and the output they produce.

● Table 28. Individual Type Codes Used in SYMDMP Output.

Code	Meaning
A	Alphabetic
B	Binary
D	Display
E	Edited
*	Subscripted Item
F	Floating Point
N	Numeric
P	Packed Decimal
S	Signed
OT	Overpunch Sign Trailing
OL	Overpunch Sign Leading
SL	Separate Sign Leading
ST	Separate Sign Trailing

```
// JOB DEBUGL
// OPTION NODECK,LINK,LIST,LISTX,SYM,ERRS,DUMP
// ASSGN SYS005,X'185'  ←Ⓐ
// EXEC FCOBOL
```

```
  CBL SXREF,FLOW=10,SYMDMP,QUOTE,SEQ ←Ⓑ
00001    000010 IDENTIFICATION DIVISION.
00002    000020 PROGRAM-ID. TESTRUN.
00003    000030    AUTHOR. PROGRAMMER NAME.
00004    000040    INSTALLATION. NEW YORK PROGRAMMING CENTER.
00005    000050    DATE-WRITTEN. FEBRUARY 10, 1971.
00006    000060 DATE-COMPILED. 03/05/71
00007    000070    REMARKS. THIS PROGRAM HAS BEEN WRITTEN AS A SAMPLE PROGRAM FOR
00008    000080    COBOL USERS. IT CREATES AN OUTPUT FILE AND READS IT BACK AS
00009    000090    INPUT.
00010    000100
00011    000110 ENVIRONMENT DIVISION.
00012    000120 CONFIGURATION SECTION.
00013    000130 SOURCE-COMPUTER. IBM-360-H50.
00014    000140 OBJECT-COMPUTER. IBM-360-H50.
00015    000150 INPUT-OUTPUT SECTION.
00016    000160 FILE-CONTROL.
00017    000170    SELECT FILE-1 ASSIGN TO SYS008-UT-2400-S.
00018    000180    SELECT FILE-2 ASSIGN TO SYS008-UT-2400-S.
00019    000190
00020    000200 DATA DIVISION.
00021    000210 FILE SECTION.
00022    000220 FD   FILE-1
00023    000230    LABEL RECORDS ARE OMITTED
00024    000240    BLOCK CONTAINS 5 RECORDS
00025    000250    RECORDING MODE IS F
00026    000255    RECORD CONTAINS 20 CHARACTERS
00027    000260    DATA RECORD IS RECORD-1.
00028    000270 01  RECORD-1.
00029    000280    05 FIELD-A PIC X(20).
00030    000290 FD   FILE-2
00031    000300    LABEL RECORDS ARE OMITTED
00032    000310    BLOCK CONTAINS 5 RECORDS
00033    000320    RECORD CONTAINS 20 CHARACTERS
00034    000330    RECORDING MODE IS F
00035    000340    DATA RECORD IS RECORD-2.
00036    000350 01  RECORD-2.
00037    000360    05 FIELD-A PIC X(20).
```

● Figure 22.   Using the Symbolic Debugging Features to Debug the
              Program TESTRUN (Part 1 of 12)

```
00038    000370 WORKING-STORAGE SECTION.
00039    000380 01  FILLER.
00040    000390     02 COUNT PIC S99 COMP SYNC.
00041    000400     02 ALPHABET PIC X(26) VALUE IS "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
00042    000410     02 ALPHA REDEFINES ALPHABET PIC X OCCURS 26 TIMES.
00043    000420     02 NUMBR PIC S99 COMP SYNC.
00044    000430     02 DEPENDENTS PIC X(26) VALUE "01234012340123401234012340".
00045    000440     02 DEPEND REDEFINES DEPENDENTS PIC X OCCURS 26 TIMES.
00046    000450 01  WORK-RECORD.
00047    000460     05 NAME-FIELD PIC X.
00048    000470     05 FILLER PIC X VALUE IS SPACE.
00049    000480     05 RECORD-NO PIC 9999.
00050    000490     05 FILLER PIC X VALUE IS SPACE.
00051    000500     05 LOCATION PIC AAA VALUE IS "NYC".
00052    000510     05 FILLER PIC X VALUE IS SPACE.
00053    000520     05 NO-OF-DEPENDENTS PIC XX.
00054    000530     05 FILLER PIC X(7) VALUE IS SPACES.
00055    000534 01  RECORDA.
00056    000535     02 A PICTURE S9(4) VALUE 1234.
00057    000536     02 B REDEFINES A PICTURE S9(7) COMPUTATIONAL-3.
00058    000540
00059    000550 PROCEDURE DIVISION.
00060    000560 BEGIN.
00061    000570     NOTE THAT THE FOLLOWING OPENS THE OUTPUT FILE TO BE CREATED
00062    000580     AND INITIALIZES COUNTERS.
00063    000590 STEP-1. OPEN OUTPUT FILE-1. MOVE ZERO TO COUNT, NUMBR.
00064    000600     NOTE THAT THE FOLLOWING CREATES INTERNALLY THE RECORDS TO BE
00065    000610     CONTAINED IN THE FILE, WRITES THEM ON TAPE, AND DISPLAYS
00066    000620     THEM ON THE CONSOLE.
00067    000630 STEP-2. ADD 1 TO COUNT, NUMBR. MOVE ALPHA (COUNT) TO
00068    000640     NAME-FIELD.
00069    000650     MOVE DEPEND (COUNT) TO NO-OF-DEPENDENTS.
00070    000660     MOVE NUMBR TO RECORD-NO.
00071    000670 STEP-3. DISPLAY WORK-RECORD UPON CONSOLE. WRITE RECORD-1 FROM
00072    000680     WORK-RECORD.
00073    000690 STEP-4. PERFORM STEP-2 THRU STEP-3 UNTIL COUNT IS EQUAL TO 26.
00074    000700     NOTE THAT THE FOLLOWING CLOSES THE OUTPUT FILE AND REOPENS
00075    000710     IT AS INPUT.
00076    000720 STEP-5. CLOSE FILE-1. OPEN INPUT FILE-2.
00077    000730     NOTE THAT THE FOLLOWING READS BACK THE FILE AND SINGLES
00078    000740     OUT EMPLOYEES WITH NO DEPENDENTS.
00079    000750 STEP-6. READ FILE-2 RECORD INTO WORK-RECORD AT END GO TO STEP-8.
00080    000755     COMPUTE B = B + 1.
00081    000760 STEP-7. IF NO-OF-DEPENDENTS IS EQUAL TO "0" MOVE "Z" TO
00082    000770     NO-OF-DEPENDENTS. EXHIBIT NAMED WORK-RECORD. GO TO STEP-6.
00083    000780 STEP-8. CLOSE FILE-2.
00084    000790     STOP RUN.
```

● Figure 22. Using the Symbolic Debugging Features to Debug the
            Program TESTRUN (Part 2 of 12)

INTRNL NAME	LVL	SOURCE NAME	BASE	DISPL	INTRNL NAME	DEFINITION	USAGE	R	O	Q	M
DNM=1-148	FD	FILE-1	DTF=01		DNM=1-148		DTFMT				F
DNM=1-178	01	RECORD-1	BL=1	000	DNM=1-178	DS 0CL20	GROUP				
DNM=1-199	02	FIELD-A	BL=1	000	DNM=1-199	DS 20C	DISP				
DNM=1-216	FD	FILE-2	DTF=02		DNM=1-216		DTFMT				F
DNM=1-246	01	RECORD-2	BL=2	000	DNM=1-246	DS 0CL20	GROUP				
DNM=1-267	02	FIELD-A	BL=2	000	DNM=1-267	DS 20C	DISP				
DNM=1-287	01	FILLER	BL=3	000	DNM=1-287	DS 0CL56	GROUP				
DNM=1-306	02	COUNT	BL=3	000	DNM=1-306	DS 1H	COMP				
DNM=1-321	02	ALPHABET	BL=3	002	DNM=1-321	DS 26C	DISP				
DNM=1-339	02	ALPHA	BL=3	002	DNM=1-339	DS 1C	DISP	R	O		
DNM=1-357	02	NUMBR	BL=3	01C	DNM=1-357	DS 1H	COMP				
DNM=1-372	02	DEPENDENTS	BL=3	01E	DNM=1-372	DS 26C	DISP				
DNM=1-392	02	DEPEND	BL=3	01E	DNM=1-392	DS 1C	DISP	R	O		
DNM=1-408	01	WORK-RECORD	BL=3	038	DNM=1-408	DS 0CL20	GROUP				
DNM=1-432	02	NAME-FIELD	BL=3	038	DNM=1-432	DS 1C	DISP				
DNM=1-452	02	FILLER	BL=3	039	DNM=1-452	DS 1C	DISP				
DNM=1-471	02	RECORD-NO	BL=3	03A	DNM=1-471	DS 4C	DISP-NM				
DNM=1-490	02	FILLER	BL=3	03E	DNM=1-490	DS 1C	DISP				
DNM=2-000	02	LOCATION	BL=3	03F	DNM=2-000	DS 3C	DISP				
DNM=2-018	02	FILLER	BL=3	042	DNM=2-018	DS 1C	DISP				
DNM=2-037	02	NO-OF-DEPENDENTS	BL=3	043	DNM=2-037	DS 2C	DISP				
DNM=2-063	02	FILLER	BL=3	045	DNM=2-063	DS 7C	DISP				
DNM=2-082	01	RECORDA	BL=3	050	DNM=2-082	DS 0CL4	GROUP				
DNM=2-102	02	A	BL=3	050	DNM=2-102	DS 4C	DISP-NM				
DNM=2-113	02	B	BL=3	050	DNM=2-113	DS 4P	COMP-3	R			

● Figure 22.   Using the Symbolic Debugging Features to Debug the
Program TESTRUN (Part 3 of 12)

```
                MEMORY MAP

        TGT                    003E8

   SAVE AREA                   003E8
   SWITCH                      00430
   TALLY                       00434
   SORT SAVE                   00438
   ENTRY-SAVE                  0043C
   SORT CORE SIZE              00440
   NSTD-REELS                  00444
   SORT RET                    00446
   WORKING CELLS               00448
   SORT FILE SIZE              00578
   SORT MODE SIZE              0057C
   PGT-VN TBL                  00580
   TGT-VN TBL                  00584
   SORTAB ADDRESS              00588
   LENGTH OF VN TBL            0058C
   LNGTH OF SORTAB             0058E
   PGM ID                      00590
   A(INIT1)                    00598
   UPSI SWITCHES               0059C
   DEBUG TABLE PTR             005A4
   CURRENT PRIORITY            005A8
   TA LENGTH                   005A9
   PROCEDURE BLOCK1 PTR        005AC
   UNUSED                      005B0
   OVERFLOW CELLS              005B4
   BL CELLS                    005B4
   DTFADR CELLS                005C0
   TEMP STORAGE                005C8
   TEMP STORAGE-2              005D0
   TEMP STORAGE-3              005D0
   TEMP STORAGE-4              005D0
   BLL CELLS                   005D0
   VLC CELLS                   005D4
   SBL CELLS                   005D4
   INDEX CELLS                 005D4
   SUBADR CELLS                005D4
   ONCTL CELLS                 005DC
   PFMCTL CELLS                005DC
   PFMSAV CELLS                005DC
   VN CELLS                    005E0
   SAVE AREA =2                005E4
   XSASW CELLS                 005E4
   XSA CELLS                   005E4
   PARAM CELLS                 005E4
   RPTSAV AREA                 005E8
   CHECKPT CTR                 005E8
   IOPTR CELLS                 005E8
   DEBUG TABLE                 005E8
```

● Figure 22. Using the Symbolic Debugging Features to Debug the
              Program TESTRUN (Part 4 of 12)

```
LITERAL POOL (HEX)

00650 (LIT+0)      00000001  001A1C5B  5BC2D6D7  C5D5405B  5BC2C3D3  D6E2C55B
00668 (LIT+24)     5BC2C6C3  D4E4D300  C0000000

      DISPLAY LITERALS (BCD)

00674 (LTL+36)    'WORK-RECORD'


                  PGT                        005F8

          DEBUG LINKAGE AREA               005F8
          OVERFLOW CELLS                   00600
          VIRTUAL CELLS                    00600
          PROCEDURE NAME CELLS             00624
          GENERATED NAME CELLS             00634
          SUBDTF ADDRESS CELLS             00648
          VNI CELLS                        00648
          LITERALS                         00650
          DISPLAY LITERALS                 00674
          PROCEDURE BLOCK CELLS            00680


    REGISTER ASSIGNMENT

    REG 6    BL =3
    REG 7    BL =1
    REG 8    BL =2

WORKING-STORAGE STARTS AT LOCATION 000E8 FOR A LENGTH OF 00058.

    60             000680                    START   EQU    *
                   000680   07 00                    BCR    0,0
                   000682   58 F0 C 018              L      15,018(0,12)        V(ILBDDBG4)
                   000686   05 EF                    BALR   14,15
                   000688   58 F0 C 01C              L      15,01C(0,12)        V(ILBDFLW1)
                   00068C   05 1F                    BALR   1,15
                   00068E   003C                     DC     X'003C'
    63             000690   07 00                    BCR    0,0
                   000692   58 F0 C 018              L      15,018(0,12)        V(ILBDDBG4)
                   000696   05 EF                    BALR   14,15
                   000698   58 F0 C 01C              L      15,01C(0,12)        V(ILBDFLW1)
                   00069C   05 1F                    BALR   1,15
                   00069E   003F                     DC     X'003F'
    63             0006A0   58 F0 C 018              L      15,018(0,12)        V(ILBDDBG4)
                   0006A4   05 EF                    BALR   14,15
                   0006A6   41 10 C 05F              LA     1,05F(0,12)         LIT+7
                   0006AA   58 00 D 1D8              L      0,1D8(0,13)         DTF=1
                   0006AE   18 40                    LR     4,0
                   0006B0   07 00                    BCR    0,0
                   0006B2   05 F0                    BALR   15,0
                   0006B4   50 00 F 008              ST     0,008(0,15)
                   0006B8   45 00 F 00C              BAL    0,00C(0,15)
                   0006BC   00000000                 DC     X'00000000'
                   0006C0   0A 02                    SVC    2
```

● Figure 22.   Using the Symbolic Debugging Features to Debug the
               Program TESTRUN (Part 5 of 12)

Ⓒ   CROSS-REFERENCE DICTIONARY

DATA NAMES   DEFN   REFERENCE

A   000056
ALPHA   000042   000067
ALPHABET   000041
B   000057   000080
COUNT   000040   000063   000067   000069   000073
DEPEND   000045   000069
DEPENDENTS   000044
FIELD-A   000029
FIELD-A   000037
FILE-1   000017   000063   000071   000076
FILE-2   000018   000076   000079   000083
LOCATION   000051
NAME-FIELD   000047   000067
NO-OF-DEPENDENTS   000053   000069   000081
NUMBR   000043   000063   000067   000070
RECORD-NO   000049   000070
RECORD-1   000028   000071
RECORD-2   000036   000079
RECORDA   000055
WORK-RECORD   000046   000071   000079   000082

Ⓒ
PROCEDURE NAMES   DEFN   REFERENCE

BEGIN   000060
STEP-1   000063
STEP-2   000067   000073
STEP-3   000071   000073
STEP-4   000073
STEP-5   000076
STEP-6   000079   000082
STEP-7   000081
STEP-8   000083   000079

CARD   ERROR MESSAGE

56   ILA2190I-W   PICTURE CLAUSE IS SIGNED, VALUE CLAUSE UNSIGNED. ASSUMED POSITIVE.
67   ILA5011I-W   HIGH ORDER TRUNCATION MIGHT OCCUR.
67   ILA5011I-W   HIGH ORDER TRUNCATION MIGHT OCCUR.

// EXEC LNKEDT

JOB   DEBUGL   DISK LINKAGE EDITOR DIAGNOSTIC OF INPUT

ACTION TAKEN   MAP
LIST   INCLUDE ILBDDBG0
LIST   AUTOLINK   IJFFBZZN
LIST   AUTOLINK   IJJCPDV1
LIST   AUTOLINK   ILBDADR0
LIST   AUTOLINK   ILBDDSP0
LIST   INCLUDE IJJCPD1
LIST   AUTOLINK   ILBDFLW0
LIST   AUTOLINK   ILBDIML0
LIST   AUTOLINK   ILBDNNS0
LIST   AUTOLINK   ILBDSAE0
LIST   ENTRY

● Figure 22.   Using the Symbolic Debugging Features to Debug the
               Program TESTRUN (Part 6 of 12)

PHASE	XFR-AD	LOCORE	HICORE	DSK-AD	ESD TYPE	LABEL	LOADED	REL-FR
PHASE***	003CE0	0032A0	005E2B	66 04 2	CSECT	ILBDDBG0	0032A0	0032A0
					* ENTRY	ILBDDBG1	0033F6	
					ENTRY	ILBDDBG2	003564	
					* ENTRY	ILBDDBG3	0037D2	
					ENTRY	ILBDDBG4	0037DC	
					ENTRY	ILBDDBG5	00376A	
					* ENTRY	ILBDDBG6	0037EC	
					ENTRY	STXITPSW	003898	
					ENTRY	ILBDDBG7	003800	
					* ENTRY	SORTEP	003A38	
					CSECT	ILBDFLW0	005818	005818
					ENTRY	ILBDFLW2	0059B4	
					ENTRY	ILBDFLW1	0058D8	
					CSECT	ILBDDSP0	005038	005038
					* ENTRY	ILBDDSS0	005038	
					* ENTRY	ILBDDSS1	005590	
					* ENTRY	ILBDDSS2	005628	
					* ENTRY	ILBDDSS3	0057E0	
					CSECT	IJJCPDV1	004B00	004B00
					* ENTRY	IJJCPDV2	004B00	
					CSECT	TESTRUN	003CE0	003CE0
					CSECT	IJFFBZZN	004790	004790
					* ENTRY	IJFFZZZN	004790	
					* ENTRY	IJFFBZZZ	004790	
					* ENTRY	IJFFZZZZ	004790	
					CSECT	ILBDSAE0	005D10	005D10
					ENTRY	ILBDSAE1	005D30	
					CSECT	ILBDMNS0	005D08	005D08
					CSECT	ILBDIML0	005CB0	005CB0
					CSECT	ILBDADR0	004CE0	004CE0
					* ENTRY	ILBDADR1	004CEC	
					CSECT	IJJCPD1	004E40	004E40
					ENTRY	IJJCPD1N	004E40	
					* ENTRY	IJJCPD3	004E40	

* UNREFERENCED SYMBOLS

EXTRN   ILBDSTN0
EXTRN   ILBDSRT0
EXTRN   ILBDTEF3

003 UNRESOLVED ADDRESS CONSTANTS

```
// ASSGN SYS008,X'182'
// ASSGN SYS009,X'185' ──(D)
// EXEC
```

● Figure 22.   Using the Symbolic Debugging Features to Debug the
               Program TESTRUN (Part 7 of 12)

(1) TESTRUN,009,MT,(HEX)

(2) 71,ON 1,4,(HEX),COUNT,NAME-FIELD,NO-OF-DEPENDENTS,RECORD-NO

(3) 80,(HEX),WORK-RECORD,B

} (E)

NO ERRORS FOUND IN CONTROL CARDS

(F)

TYPE CODES USED IN SYMDMP OUTPUT

CODE	MEANING
A	= ALPHABETIC
AN	= ALPHANUMERIC
ANE	= ALPHANUMERIC EDITED
D	= DISPLAY (STERLING NONREPORT)
DE	= DISPLAY EDITED (STERLING REPORT)
F	= FLOATING POINT (COMP-1/COMP-2)
FD	= FLOATING POINT DISPLAY (EXTERNAL FLOATING POINT)
NB	= NUMERIC BINARY UNSIGNED (COMP)
NB-S	= NUMERIC BINARY SIGNED
ND	= NUMERIC DISPLAY UNSIGNED (EXTERNAL DECIMAL)
ND-OL	= NUMERIC DISPLAY OVERPUNCH SIGN LEADING
ND-OT	= NUMERIC DISPLAY OVERPUNCH SIGN TRAILING
ND-SL	= NUMERIC DISPLAY SEPARATE SIGN LEADING
ND-ST	= NUMERIC DISPLAY SEPARATE SIGN TRAILING
NE	= NUMERIC EDITED
NP	= NUMERIC PACKED DECIMAL UNSIGNED (COMP-3)
NP-S	= NUMERIC PACKED DECIMAL SIGNED
*	= SUBSCRIPTED

(G)

```
TESTRUN  AT CARD 000071
  LOC     CARD   LV NAME                          TYPE    VALUE

003DC8  000040  02 COUNT                          NB-S    +01
                                          (HEX)           0001

003E00  000047  02 NAME-FIELD                     AN      A

003E0B  000053  02 NO-OF-DEPENDENTS               AN      0

003E02  000049  02 RECORD-NO                      ND      0001


TESTRUN  AT CARD 000071
  LOC     CARD   LV NAME                          TYPE    VALUE

003DC8  000040  02 COUNT                          NB-S    +05
                                          (HEX)           0005

003E00  000047  02 NAME-FIELD                     AN      E

003E0B  000053  02 NO-OF-DEPENDENTS               AN      4

003E02  000049  02 RECORD-NO                      ND      0005


TESTRUN  AT CARD 000071
  LOC     CARD   LV NAME                          TYPE    VALUE

003DC8  000040  02 COUNT                          NB-S    +09
                                          (HEX)           0009

003E00  000047  02 NAME-FIELD                     AN      I

003E0B  000053  02 NO-OF-DEPENDENTS               AN      3

003E02  000049  02 RECORD-NO                      ND      0009
```

● Figure 22.   Using the Symbolic Debugging Features to Debug the
                Program TESTRUN (Part 8 of 12)

```
TESTRUN   AT CARD 000071
 LOC      CARD   LV NAME                              TYPE    VALUE

003DC8   000040  02 COUNT                             NB-S    +13
                                              (HEX)           000D

003E00   000047  02 NAME-FIELD                        AN      M

003E0B   000053  02 NO-OF-DEPENDENTS                  AN      2

003E02   000049  02 RECORD-NO                         ND      0013


TESTRUN   AT CARD 000071
 LOC      CARD   LV NAME                              TYPE    VALUE

003DC8   000040  02 COUNT                             NB-S    +17
                                              (HEX)           0011

003E00   000047  02 NAME-FIELD                        AN      Q

003E0B   000053  02 NO-OF-DEPENDENTS                  AN      1

003E02   000049  02 RECORD-NO                         ND      0017


TESTRUN   AT CARD 000071
 LOC      CARD   LV NAME                              TYPE    VALUE

003DC8   000040  02 COUNT                             NB-S    +21
                                              (HEX)           0015

003E00   000047  02 NAME-FIELD                        AN      U

003E0B   000053  02 NO-OF-DEPENDENTS                  AN      0

003E02   000049  02 RECORD-NO                         ND      0021


TESTRUN   AT CARD 000071
 LOC      CARD   LV NAME                              TYPE    VALUE

003DC8   000040  02 COUNT                             NB-S    +25
                                              (HEX)           0019

003E00   000047  02 NAME-FIELD                        AN      Y

003E0B   000053  02 NO-OF-DEPENDENTS                  AN      4

003E02   000049  02 RECORD-NO                         ND      0025
```

(H)

```
TESTRUN   AT CARD 000080
 LOC      CARD   LV NAME                              TYPE    VALUE

         000046  01 WORK-RECORD
003E00                                        (HEX)           C140F0F0 F0F140D5 E8C340F0   40404040 40404040
003E00   000047  02 NAME-FIELD                        AN      A
003E01   000048  02 FILLER                            AN
003E02   000049  02 RECORD-NO                         ND      0001
003E06   000050  02 FILLER                            AN
003E07   000051  02 LOCATION                          A       NYC
003E0A   000052  02 FILLER                            AN
003E0B   000053  02 NO-OF-DEPENDENTS                  AN      0
003E0D   000054  02 FILLER                            AN

003E18   000057  02 B                                 NP-S    *1*2*3*
                                              (HEX)           F1F2F3C4
```

• Figure 22.   Using the Symbolic Debugging Features to Debug the
               Program TESTRUN (Part 9 of 12)

COBOL ABEND DIAGNOSTIC AIDS

INTERRUPT CODE 7   LAST PSW ADDR BEFORE ABEND C00045DC ◄——— (I)

PROGRAM TESTRUN

LAST CARD NUMBER/VERB NUMBER EXECUTED -- CARD NUMBER 000080/VERB NUMBER 01.   } (J)

FLOW TRACE
TESTRUN 000067 000071 000067 000071 000067 000071 000067 000071 000076 000079   (K)

DATA DIVISION DUMP OF TESTRUN

(L)

TASK GLOBAL TABLE	LOC	VALUE							
SAVE AREA	0040C8	005E9103	102D4780	00005C10	800045BC	000045D6	400045D8	00003EB0	00003FF8
	0040E8	0000001A	00003EB0	500046CC	00003DC8	00003F3C	00003FF8	00004696	00003CE0
	004108	400045DC	000042D8						
SWITCH	004110	3C00004B							
TALLY	004114	00000000							
SORT-SAVE	004118	00000000							
ENTRY-SAVE	00411C	00004360							
SORT-CORE-SIZE	004120	00000000							
NSTD-REELS	004124	9508							
SORT-RETURN	004126	4000							
WORKING CELLS	004128	00003EB0	00003F90	0000001A	00003EB0	500046CC	F41E9101	102C4710	E940F0F0
	004148	F2F640D5	E8C340F0	40404040	40404040	F3DE9604	10595820	1028D201	20001058
	004168	D4012000	F43E9104	10154780	F3FC9400	102D07FE	4400F0CA	47F0F3F6	00FFFFFF
	004188	FFFF9601	10104400	F01C4580	F0D694BE	101047F0	F0C29620	10599608	105947F0
	0041A8	F3D21309	04130504	02190216	00080000	01000000	A71849FF	00000078	000044AC
	0041C8	0000001A	01004740	0000449A	00005038	50004488	000044AE	0000001A	0000001A
	0041E8	00003DFF	500046CC	00003DC8	00003F28	00003FF8	00004696	00003CE0	5000448A
	004208	000042D8	F3F29001	D0781804	181545F0	F0229001	D0781802	00029400	00004140
	004228	D08245F0	F0189400	D0821200	47B0F00E	9602D082	5D00F03E	4E1D0068	4E0D0070
	004248	F157D060	D068D204	D065D073	9102D082				
SORT-FILE-SIZE	004258	00000000							
SORT-MODE-SIZE	00425C	00000000							
PGT-VN TBL	004260	9101D082							
TGT-VN TBL	004264	078E9801							
SORTAB ADDR	004268	D07807FE							
VN TBL LENGTH	00426C	3B9A							
SORTAB LENGTH	00426E	CA00							
PROGRAM-ID	004270	TESTRUN							
A(INIT1)	004278	00003CE0							
UPSI-SWITCHES	00427C	47F0F02A	D7011052						
TGT-DBG TABLE	004284	00000200							
CURRENT PRIORITY	004288	00							
TRANSIENT AREA LENGTH	004289	CC4530							
PROCEDURE-BLOCK	00428C	F13C9826							
UNUSED	004290	F1CC9104							
OVERFLOW CELLS	(NONE)								
BL CELLS	004294	00003F3C	00003FF8	00003DC8					
DTFADR CELLS	0042A0	00003E38	00003EB0						
TEMP STORAGE	0042A8	00000000	0000026C						
BLL CELLS	0042B0	00000000							
VLC CELLS	(NONE)								
SBL CELLS	(NONE)								
INDEX CELLS	(NONE)								
OTHER (SEE MEMORY MAP)	0042B4	00003DE3	00003DFF	000044D6	000044D6	00004140	0A000A2C	0000099C	5482F068
	0042D4	4640F05A							

● Figure 22.   Using the Symbolic Debugging Features to Debug the
                Program TESTRUN (Part 10 of 12)

DATA DIVISION DUMP OF TESTRUN

LOC	CARD	LV NAME		TYPE	VALUE
	000017	FD FILE-1	(M)		STANDARD SEQUENTIAL, ASSIGNED TO SYS008, CLOSED
003E20				PRE-DTF	01010014 00000000 00000000 00000000 0F000000 00000000
003E38				DTFMT	00009200 0C000108 00003E70 00003E78 10004790 1160E2E8 E2F0F0F8 40400162
003E58					00000000 00000000 00000000 86BCF018 41E0E001 58201044 01003F28 20000064
003E78					00003F90 00003F90 00000014 00003FF3 00640063 00000000 00000000 60005D10
003E98					01010014 00000000 00000000 00000000 0F000000 00000000 00008200 0C000108
	000028	01 RECORD-1	(N)		
003F3C				(HEX)	D840F0F0 F1F740D5 E8C340F1 40404040 40404040
003F3C	000029	02 FIELD-A		AN	Q 0017 NYC 1
	000018	FD FILE-2	(M)		STANDARD SEQUENTIAL, ASSIGNED TO SYS008, OPEN INPUT
003E98				PRE-DTF	01010014 00000000 00000000 00000000 0F000000 00000000
003EB0				DTFMT	00008200 0C000108 00003EE8 00003EF0 10004790 11E8E2E8 E2F0F0F8 40400272
003ED0					10000000 240045D0 00000001 86BCF018 41E0E001 58201044 02004060 00000064
003EF0					00003FF8 00003FF8 00000014 0000405B 00640063 00000000 00005D30 00005D10
003F10					00000000 00000000 00000000 00000000 96011058 47F0F3FC E940F0F0 F2F640D5
	000036	01 RECORD-2	(N)		
003FF8				(HEX)	C140F0F0 F0F140D5 E8C340F0 40404040 40404040
003FF8	000037	02 FIELD-A		AN	A 0001 NYC 0
	000039	01 FILLER	(P)		
003DC8				(HEX)	001AC1C2 C3C4C5C6 C7C8C9D1 D2D3D4D5 D6D7D8D9 E2E3E4E5
003DE0					E6E7E8E9 001AF0F1 F2F3F4F0 F1F2F3F4 F0F1F2F3 F4F0F1F2
003DF8					F3F4F0F1 F2F3F4F0
003DC8	000040	02 COUNT		NB-S	+26
003DCA	000041	02 ALPHABET		AN	ABCDEFGHIJKLMNOPQRSTUVWXYZ
	000042	02 ALPHA		*AN	

	(Q) (SUB1)		
003DCA	1	A
003DCB	2	B
003DCC	3	C
003DCD	4	D
003DCE	5	E
003DCF	6	F
003DD0	7	G
003DD1	8	H
003DD2	9	I
003DD3	10	J
003DD4	11	K
003DD5	12	L
003DD6	13	M
003DD7	14	N
003DD8	15	O
003DD9	16	P
003DDA	17	Q
003DDB	18	R
003DDC	19	S
003DDD	20	T
003DDE	21	U

● Figure 22. Using the Symbolic Debugging Features to Debug the
Program TESTRUN (Part 11 of 12)

DATA DIVISION DUMP OF TESTRUN

LOC	CARD	LV	NAME		TYPE	VALUE
003DDF				22		V
003DE0				23		W
003DE1				24		X
003DE2				25		Y
003DE3				26		Z
003DE4	000043	02	NUMBR		NB-S	+26
003DE6	000044	02	DEPENDENTS		AN	01234012340123401234012340
	000045	02	DEPEND		*AN	
			Ⓠ	(SUB1)		
003DE6				1		0
003DE7				2		1
003DE8				3		2
003DE9				4		3
003DEA				5		4
003DEB				6		0
003DEC				7		1
003DED				8		2
003DEE				9		3
003DEF				10		4
003DF0				11		0
003DF1				12		1
003DF2				13		2
003DF3				14		3
003DF4				15		4
003DF5				16		0
003DF6				17		1
003DF7				18		2
003DF8				19		3
003DF9				20		4
003DFA				21		0
003DFB				22		1
003DFC				23		2
003DFD				24		3
003DFE				25		4
003DFF				26		0

LOC	CARD	LV	NAME		TYPE	VALUE
	000046	01	WORK-RECORD	Ⓟ		
003E00					(HEX)	C140F0F0 F0F140D5 E8C340F0 40404040 40404040
003E00	000047	02	NAME-FIELD		AN	A
003E01	000048	02	FILLER		AN	
003E02	000049	02	RECORD-NO		ND	0001
003E06	000050	02	FILLER		AN	
003E07	000051	02	LOCATION		A	NYC
003E0A	000052	02	FILLER		AN	
003E0B	000053	02	NO-OF-DEPENDENTS		AN	0
003E0D	000054	02	FILLER		AN	
	000055	01	RECORDA	Ⓟ		
003E18					(HEX)	F1F2F3C4
003E18	000056	02	A		ND-OT	+1234
003E18	000057	02	B	Ⓡ→	NP-S	*1*2*3*

END OF COBOL DIAGNOSTIC AIDS

● Figure 22.   Using the Symbolic Debugging Features to Debug the
             Program TESTRUN (Part 12 of 12)

Program Product Information -- Version 3

APPENDIX G: 3525 COMBINED FUNCTION PROCESSING

The IBM 80-column card punch device offers more flexible
processing capabilities than former card devices. When equipped
with appropriate special features, the 3525 can be used separately
as a card reader, as a card punch, or as a card printer. (Either
2-line printing or 25-line printing is available.) Any two or all
three of these functions can be combined, so that those functions
specified are all performed during one pass of a card through the
device.

For any one data card, the operations, when specified, must be
performed in the following order: read, punch, print. Any one
function may be omitted -- that is, no file need be defined for that
function. The remaining functions must still be performed in the
order shown. All operations on one card must be completed before
operations on the next card are begun, or there is an abnormal
termination of the job. When such combined function processing is
to be used, the programmer must be aware of the special
considerations needed to accomplish the desired results.

COBOL handles each of the separate functions to be combined as a
separate logical file. Each such logical file has its own file
structure and procedural processing requirements. However, because
such combined function files refer to one physical unit, the user
must observe certain restrictions during processing. The following
sections explain the programming requirements for combined function
processing in DOS American National Standard COBOL.

I -- ENVIRONMENT DIVISION CONSIDERATIONS

For combined function processing there are special considerations
for the SPECIAL-NAMES Paragraph, and for the SELECT, ASSIGN, and
RESERVE Clauses.

SPECIAL-NAMES Paragraph

If stacker selection of punched output is desired, mnemonic-names
for the purpose can be specified in the SPECIAL-NAMES Paragraph.
The mnemonic-names may be equated with the following function-names:

Function-name	Meaning
S01	Stacker 1
S02	Stacker 2

SELECT Clause

For each of the functions (reading, punching, printing) to be
combined, a unique file-name must be defined.

ASSIGN Clause

For combined function card files, the ASSIGN clause has the
following format:

$$
SYSnnn-3525
\begin{Bmatrix} R \\ P \\ W \\ M \end{Bmatrix}
-
\begin{Bmatrix} X[R] \\ Y[R] \\ V[R] \\ Z \end{Bmatrix}
[-name]
$$

where nnn is a 3-digit number between 000 and 221, inclusive. Each
of the associated logical files must be specified with the same
SYSnnn field. (The field represents the symbolic unit to which the
logical file is assigned.) The name field has the same meaning it
has for other files.

The device and organization fields are interdependent. The
following entries are valid:

Device	Organization	Meaning
3525R (reader)	V[R]	read/print associated file
	X[R]	read/punch/print associated file

Note: The optional R code in the
organization field specifies RCE
(Read Column Eliminate) card reading.

Device	Organization	Meaning
3525P (punch)	X	read/punch/print associated file
	Y	read/punch associated file
	Z	punch/print associated file
3525W (2-line printer)	V	read/print associated file
	X	read/punch/print associated file
	Z	punch/print associated file
3525M (multiline printer)	V	read/print associated file
	X	read/punch/print associated file
	Z	punch/print associated file

RESERVE Clause

For a combined function structure, the files assigned to the read
function and the punch function must each specify RESERVE NO
ALTERNATE AREA(S).

A file assigned to the print function may specify either NO or 1
ALTERNATE AREA(S).

II -- DATA DIVISION CONSIDERATIONS

For each logical file defined in the Environment Division for the
combined function structure, there must be a corresponding FD entry
and 01 record description entry in the File Section of the Data
Division.

III -- PROCEDURE DIVISION CONSIDERATIONS

When combined function processing is to be used, input/output operations must proceed in a specified order in the Procedure Division. In the 3525 device, the card passes first through the reading station, next through the punching station, and last through the printing station. Therefore, the following combined functions may be specified, in the order shown:

Functions to be Combined	Order of Operations	Associated COBOL Statement
read/punch/print	read punch [print]	READ ... AT END WRITE ... [ADVANCING/POSITIONING] WRITE
read/punch	read punch	READ ... AT END WRITE ... [ADVANCING/POSITIONING]
read/print	read [print]	READ ... AT END WRITE
punch/print	punch [print]	WRITE ... [ADVANCING/POSITIONING] WRITE

All operations on one card must be completed before the next card is obtained, or there is an abnormal termination of the job.

The following Procedure Division considerations in the COBOL source program apply:

OPEN Statement

Combined function files may be opened in any sequence. The read function file must be opened INPUT; the punch and print function files must be opened OUTPUT. All files must be opened before processing begins; if they are not, the job is terminated.

READ Statement

For combined function files, the READ statement, if the function is specified, must be the first input/output operation specified. A second READ statement must not be issued before all necessary combined function operations for the same card have been completed, or abnormal termination of the job results.

WRITE Statement -- Punch Function Files

When the punch function is used, then after the READ statement is issued, the next input/output operation must be a WRITE statement for the punch function file.

If the user wishes to punch additional data into some of the cards and not into others, he must issue a dummy WRITE statement for the null cards, first filling the output area with SPACES.

If stacker selection for the punch function file is desired, the user can specify S01 (for logical stacker one) and S02 (for logical

stacker two) as function-names in the SPECIAL-NAMES Paragraph. He can then issue WRITE ADVANCING statements using the associated mnemonic-names. Alternatively, if he specifies WRITE AFTER POSITIONING, he must use the identifier-2 option; the values placed in identifier-2 before the statement is issued must be V (for pocket 1) or W (for pocket 2). Stacker selection may be specified only for the punch function file.

WRITE Statement -- Print Function Files

After the punch function operations (if specified) are completed, the user can issue WRITE statement(s) for the print function file.

If the user wishes to print additional data on some of the data cards and not on others, he may omit the WRITE statement for the null cards.

For a 2-line print file, the lines are printed on line 1 (top edge of card) and line 3 (between rows 11 and 12). Up to 64 characters may be printed on each line.

For a multiline print file up to 25 lines of characters may be printed. Up to 64 characters may be printed on each line.

Line control may not be specified for the print function file; automatic single spacing is provided.

Note: This means the ADVANCING/POSITIONING options may not be specified for the print function file.

Any attempt to write beyond the limits of the card results in abnormal termination of the job.

CLOSE Statement

When processing is completed, a CLOSE statement must be issued for each of the combined function files. After a CLOSE statement has been issued for any one of the functions, an attempt to perform processing for any of the functions results in abnormal termination.

ACCESS: The manner in which files are referenced by the computer.
Access can be sequential (records are referred to one after another in
the order in which they appear on the file), or it can be random (the
individual records can be referred to in a nonsequential manner).

Actual Decimal Point: The physical representation, using either of the
decimal point characters (. or ,), of the decimal point position in a
data item.   When specified, it will appear in a printed report, and it
requires an actual space in storage.

ACTUAL KEY: A key which can be directly used by the system to locate a
logical record on a mass storage device.

Alphabetic Character: A character which is one of the 26 characters of
the alphabet, or a space.   In COBOL, the term does not include any other
characters.

Alphanumeric Character: Any character in the computer's character set.

Alphanumeric Edited Character: A character within an alphanumeric
character string which contains at least one B or 0.

Arithmetic Expression: A statement containing any combination of data-
names, numeric literals, and figurative constants, joined together by
one or more arithmetic operators in such a way that the statement as a
whole can be reduced to a single numeric value.

Arithmetic Operator: A symbol (single character or two-character set)
which directs the system to perform an arithmetic operation.   The
following list shows arithmetic operators:

Meaning	Symbol
Addition	+
Subtraction	–
Multiplication	*
Division	/
Exponentiation	**

Assumed Decimal Point: A decimal point position which does not involve
the existence of an actual character in a data item.   It does not occupy
an actual space in storage, but is used by the compiler to align a value
properly for calculation.

BLOCK: In COBOL, a group of characters or records which is treated as
an entity when moved into or out of the computer.   The term is
synonymous with the term Physical Record.

Buffer: A portion of main storage into which data is read or from which
it is written.

Byte: A sequence of eight adjacent binary bits.   When properly aligned,
two bytes form a halfword, four bytes a fullword, and eight bytes a
doubleword.

Channel: A device that directs the flow of information between the
computer main storage and the input/output devices.

Character: One of a set of indivisible symbols that can be arranged in
sequences to express information.   These symbols include the letters A
through Z, the decimal digits 0 through 9, punctuation symbols, and any
other symbols which will be accepted by the data-processing system.

<u>Character Set</u>: All the valid COBOL characters. The complete set of 51 characters is listed in "Language Considerations."

<u>Character String</u>: A connected sequence of characters. All COBOL characters are valid.

<u>Checkpoint</u>: A reference point in a program at which information about the contents of core storage can be recorded so that, if necessary, the program can be restarted at an intermediate point.

<u>Class Condition</u>: A statement that the content of an item is wholly alphabetic or wholly numeric. It may be true or false.

<u>Clause</u>: A set of consecutive COBOL words whose purpose is to specify an attribute of an entry. There are three types of clauses: data, environment, and file.

<u>COBOL Character</u>: Any of the 51 valid characters (see CHARACTER) in the COBOL character set. The complete set is listed in "Language Considerations."

<u>Collating Sequence</u>: The arrangement of all valid characters in the order of their relative precedence. The collating sequence of a computer is part of the computer design -- each acceptable character has a predetermined place in the sequence. A collating sequence is used primarily in comparison operations.

<u>COLUMN Clause</u>: A COBOL clause used to identify a specific position within a report line.

<u>Comment</u>: An annotation in the Identification Division or Procedure Division of a COBOL source program. A comment is ignored by the compiler. As an IBM extension, comments may be included at any point in a COBOL source program.

<u>Compile Time</u>: The time during which a COBOL source program is translated by the COBOL compiler into a machine language object program.

<u>Compiler</u>: A program which translates a program written in a higher level language into a machine language object program.

<u>Compiler Directing Statement</u>: A COBOL statement which causes the compiler to take a specific action at compile time, rather than the object program to take a particular action at execution time.

<u>Compound Condition</u>: A statement that tests two or more relational expressions. It may be true or false.

<u>Condition</u>:

- One of a set of specified values a data item can assume.

- A simple conditional expression: relation condition, class condition, condition-name condition, sign condition, switch-status condition, NOT condition.

<u>Conditional Statement</u>: A syntactically correct statement, made up of data names, and/or literals, and/or figurative constants, and/or logical operators, so constructed that it tests a truth value. The subsequent action of the object program is dependent on this truth value.

<u>Conditional Variable</u>: A data item that can assume more than one value; one or more of the values it assumes has a condition-name assigned to it.

<u>Condition Name</u>: The name assigned to a specific value, set of values, or range of values, that a data item may assume.

Condition-name Condition: A statement that the value of a conditional variable is one of a set (or range) of values of a data item identified by a condition-name. The statement may be true or false.

CONFIGURATION SECTION: A section of the Environment Division of the COBOL program. It describes the overall specifications of computers.

Connective: A word or a punctuation character that does one of the following:

- Associates a data-name or paragraph-name with its qualifier

- Links two or more operands in a series

- Forms a conditional expression

CONSOLE: A COBOL mnemonic-name associated with the console typewriter.

Contiguous Items: Consecutive elementary or group items in the Data Division that have a definite relationship with each other.

Control Break: A recognition of a change in the contents of a control data item that governs a hierarchy.

Control Bytes: Bytes associated with a physical record that serve to identify the record and indicate its length, blocking factor, etc.

Control Data Item: A data item that is tested each time a report line is to be printed. If the value of the data item has changed, a control break occurs and special actions are performed before the line is printed.

CONTROL FOOTING: A report group that occurs at the end of the control group of which it is a member.

Control Group: An integral set of related data that is specifically associated with a control data item .

CONTROL HEADING: A report group that occurs at the beginning of the control group of which it is a member.

Control Hierarchy: A designated order of specific control data items. The highest level is the final control; the lowest level is the minor control.

Core Storage: Storage within the central processing unit of the computer, so called because this storage exists in the form of magnetic cores.

Cylinder Index: A higher level index, always present in indexed data organization. Its entries point to track indexes.

Data Description Entry: An entry in the Data Division that is used to describe the characteristics of a data item. It consists of a level number, followed by an optional data-name, followed by data clauses that fully describe the format the data will take. An elementary data description entry (or item) cannot logically be subdivided further. A group data description entry (or item) is made up of a number of related group and/or elementary items.

DATA DIVISION: One of the four main component parts of a COBOL program. The Data Division describes the files to be used in the program and the records contained within the files. It also describes any internal Working-Storage records that will be needed (see "Data Division" for full details).

Data Item: A unit of recorded information that can be identified by a symbolic name or by a combination of names and subscripts.   Elementary data items cannot logically be subdivided.   A group data item is made up of logically related group and/or elementary items and can be a logical group within a record or can itself be a complete record.

Data-name: A name assigned by the programmer to a data item in a COBOL program.   It must contain at least one alphabetic character.

DECLARATIVES: A set of one or more compiler-directing sections written at the beginning of the Procedure Division of a COBOL program.   The first section is preceded by the header DECLARATIVES.   The last section is followed by the header END DECLARATIVES.   There are three options:

1.   Input/output label handling

2.   Input/output error-checking procedures

3.   Report Writing procedures

Each has its standard format (see "Procedure Division").

Device-number: The reference number assigned to any external device.

Digit: Any of the numerals from 0 through 9.   In COBOL, the term is not used in reference to any other symbol.

DIVISION: One of the four major portions of a COBOL program:

* IDENTIFICATION DIVISION, which names the program.

* ENVIRONMENT DIVISION, which indicates the machine equipment and equipment features to be used in the program.

* DATA DIVISION, which defines the nature and characteristics of data to be processed.

* PROCEDURE DIVISION, which consists of statements directing the processing of data in a specified manner at execution time.

Division Header: The COBOL words that indicate the beginning of a particular division of a COBOL program.   The four division headers are:

* IDENTIFICATION DIVISION.

* ENVIRONMENT DIVISION.

* DATA DIVISION.

* PROCEDURE DIVISION.

Division-name: The name of one of the four divisions of a COBOL program.

EBCDIC Character: Any one of the symbols included in the eight-bit EBCDIC (Extended Binary-Coded-Decimal Interchange Code) set.   All 51 COBOL characters are included.

Editing Character: A single character or a fixed two-character combination used to create proper formats for output reports (see "Language Considerations" for a complete list of editing characters).

Elementary Item: A data item that cannot logically be subdivided.

Entry: Any consecutive set of descriptive clauses terminated by a period, written in the Identification, Environment, or Procedure Divisions of a COBOL program.

Entry-name: A programmer-specified name that establishes an entry point into a COBOL subprogram.

ENVIRONMENT DIVISION: One of the four main component parts of a COBOL program. The Environment Division describes the computers upon which the source program is compiled and those on which the object program is executed, and provides a linkage between the logical concept of files and their records, and the physical aspects of the devices on which files are stored (see "Environment Division" for full details).

Execution Time: The time at which an object program actually performs the instructions coded in the Procedure Division, using the actual data provided.

Exponent: A number, indicating how many times another number (the base) is to be repeated as a factor. Positive exponents denote multiplication, negative exponents denote division, fractional exponents denote a root of a quantity. In COBOL, exponentiation is indicated with the symbol ** followed by the exponent.

F-mode Records: Records of a fixed length, each of which is wholly contained within a block. Blocks may contain more than one record.

Figurative Constant: A reserved word that represents a numeric value, a character, or a string of repeated values or characters. The word can be written in a COBOL program to represent the values or characters without being defined in the Data Division (see "Language Considerations" for a complete list).

FILE-CONTROL: The name and Header of an Environment Division paragraph in which the data files for a given source program are named and assigned to specific input/output devices.

File Description: An entry in the File Section of the Data Division that provides information about the identification and physical structure of a file.

File-name: A name assigned to a set of input data or output data. A file-name must include at least one alphabetic character.

FILE SECTION: A section of the Data Division that contains descriptions of all externally stored data (or files) used in a program. Such information is given in one or more file description entries.

Floating-Point Literal: A numeric literal whose value is expressed in floating-point notation -- that is, as a decimal number followed by an exponent which indicates the actual placement of the decimal point.

Function-name: A name, supplied by IBM, that identifies system logical units, printer and card punch control characters, and report codes. When a function-name is associated with a mnemonic-name in the Environment Division, the mnemonic-name can then be substituted in any format in which substitution is valid.

Group Item: A data item made up of a series of logically related elementary items. It can be part of a record or a complete record.

Header Label: A record that identifies the beginning of a physical file or a volume.

High-Order : The leftmost position in a string of characters.

IDENTIFICATION DIVISION: One of the four main component parts of a COBOL program. The Identification Division identifies the source program and the object program and, in addition, may include such documentation as the author's name, the installation where written, date written, etc. (see "Identification Division" for full details).

Identifier: A data-name, unique in itself, or made unique by the syntactically correct combination of qualifiers, subscripts, and/or indexes.

Imperative-Statement: A statement consisting of an imperative verb and its operands, which specifies that an action be taken, unconditionally. An imperative-statement may consist of a series of imperative-statements.

Index: A computer storage position or register, the contents of which identify a particular element in a table.

Index Data Item: A data item in which the contents of an index can be stored without conversion to subscript form.

Index-name: A name, given by the programmer, for an index of a specific table. An index-name must contain at least one alphabetic character. It is one word (4 bytes) in length.

Indexed Data-name: A data-name identifier which is subscripted with one or more index-names.

INPUT-OUTPUT SECTION: In the Environment Division, the section that names the files and external media needed by an object program. It also provides information required for the transmission and handling of data during the execution of an object program.

INPUT PROCEDURE: A set of statements that is executed each time a record is released to the sort file. Input procedures are optional; whether they are used or not depends upon the logic of the program.

Integer: A numeric data item or literal that does not include any character positions to the right of the decimal point, actual or assumed. Where the term "integer" appears in formats, "integer" must not be a numeric data item.

INVALID KEY Condition: A condition that may arise at execution time in which the value of a specific key associated with a mass storage file does not result in a correct reference to the file (see the READ, REWRITE, START, and WRITE statements for the specific error conditions involved).

I-O-CONTROL: The name, and the header, for an Environment Division paragraph in which object program requirements for specific input/output techniques are specified. These techniques include rerun checkpoints, sharing of same areas by several data files, and multiple file storage on a single tape device.

KEY: One or more data items, the contents of which identify the type or the location of a record, or the ordering of data.

Key Word: A reserved word whose employment is essential to the meaning and structure of a COBOL statement. In this manual, key words are indicated in the formats of statements by underscoring. Key words are included in the reserved word list.

Level Indicator: Two alphabetic characters that identify a specific type of file, or the highest position in a hierarchy.   The level indicators are: FD, SD, RD.

Level Number: A numeric character or two-character set that identifies the properties of a data description entry.   Level numbers 01 through 49 define group items, the highest level being identified as 01, and the subordinate data items within the hierarchy being identified with level numbers 02 through 49.   Level numbers 66, 77, and 88 identify special properties of a data description entry in the Data Division.

Library-name: The name of a member of a data set containing COBOL entries, used with the COPY and BASIS statements.

LINKAGE SECTION: A section of the Data Division that describes data made available from another program.

Literal: A character string whose value is implicit in the characters themselves.   The numeric literal 7 expresses the value 7, and the nonnumeric literal "CHARACTERS" expresses the value CHARACTERS.

Logical Operator: A COBOL word that defines the logical connections between relational operators.   The three logical operators and their meanings are:

    OR (logical inclusive -- either or both)

    AND (logical connective -- both)

    NOT (logical negation)

(See "Procedure Division" for a more detailed explanation.)

Logical Record: The most inclusive data item, identified by a level-01 entry.   It consists of one or more related data items.

Low-Order: The rightmost position in a string of characters.

Main Program: The highest level COBOL program involved in a step. (Programs written in other languages that follow COBOL linkage conventions are considered COBOL programs in this sense.)

Mantissa: The decimal part of a logarithm.   Therefore, the part of a floating-point number that is expressed as a decimal fraction.

Mass Storage: A storage medium -- disk, drum, or data cell -- in which data can be collected and maintained in a sequential, direct, or indexed organization.

Mass Storage File: A collection of records assigned to a mass storage device.

Mass Storage File Segment: A part of a mass storage file whose beginning and end are defined by the FILE-LIMIT clause in the Environment Division.

Master Index: The highest level index, which is optional, in the indexed data organization.

Mnemonic-name: A programmer-supplied word associated with a specific function-name in the Environment Division.   It then may be written in place of the function-name in any format where such a substitution is valid.

MODE: The manner in which records of a file are accessed or processed.

Name: A word composed of not more than 30 characters, which defines a
COBOL operand (see "Language Considerations" for a more complete
discussion).

Noncontiguous Item: A data item in the Working-Storage Section of the
Data Division which bears no relationship with other data items.

Nonnumeric Literal: A character string bounded by quotation marks,
which means literally itself. For example, "CHARACTER" is the literal
for and means CHARACTER. The string of characters may include any
characters in the computer's set, with the exception of the quotation
mark. Characters that are not COBOL characters may be included.

Numeric Character: A character that belongs to one of the set of digits
0 through 9.

Numeric Edited Character: A numeric character which is in such a form
that it may be used in a printed output. It may consist of external
decimal digits 0 through 9, the decimal point, commas, the dollar sign,
etc., as the programmer wishes (see "Data Division" for a fuller
explanation).

Numeric Item: An item whose description restricts its contents to a
value represented by characters from the digits 0 through 9. The item
may also contain a leading or trailing operational sign represented
either as an overpunch or as a separate character.

Numeric Literal: A numeric character or string of characters whose
value is implicit in the characters themselves. Thus, 777 is the
literal as well as the value of the number 777.

OBJECT-COMPUTER: The name of an Environment Division paragraph in which
the computer upon which the object program will be run is described.

Object Program: The set of machine language instructions that is the
output from the compilation of a COBOL source program. The actual
processing of data is done by the object program.

Object Time: The time during which an object program is executed.

Operand: The "object" of a verb or an operator. That is, the data or
equipment governed or directed by a verb or operator.

Operational Sign: An algebraic sign associated with a numeric data
item, which indicates whether the item is positive or negative.

Optional Word: A reserved word included in a specific format only to
improve the readability of a COBOL statement. If the programmer wishes,
optional words may be omitted.

OUTPUT PROCEDURE: A set of programmer-defined statements that is
executed each time a sorted record is returned from the sort file.
Output procedures are optional; whether they are used or not depends
upon the logic of the program.

Overlay: The technique of repeatedly using the same areas of internal
storage during different stages in processing a problem.

PAGE: A physical separation of continuous data in a report. The
separation is based on internal requirements and/or the physical
characteristics of the reporting medium.

PAGE FOOTING: A report group at the end of a report page which is
printed before a page control break is executed.

PAGE HEADING: A report group printed at the beginning of a report page,
after a page control break is executed.

<u>Paragraph</u>: A set of one or more COBOL sentences, making up a logical processing entity, and preceded by a paragraph-name or a paragraph header.

<u>Paragraph Header</u>: A word followed by a period that identifies and precedes all paragraphs in the Identification Division and Environment Division.

<u>Paragraph-name</u>: A programmer-defined word that identifies and precedes a paragraph.

<u>Parameter</u>: A variable that is given a specific value for a specific purpose or process. In COBOL, parameters are most often used to pass data values between calling and called programs.

Physical Record: A physical unit of data, synonymous with a block. It can be composed of a portion of one logical record, of one complete logical record, or of a group of logical records.

Print Group: An integral set of related data within a report.

Priority-Number: A number, ranging in value from 0 to 99, which classifies source program sections in the Procedure Division (see "Segmentation" for more information).

Procedure: One or more logically connected paragraphs or sections within the Procedure Division, which direct the computer to perform some action or series of related actions.

PROCEDURE DIVISION: One of the four main component parts of a COBOL program. The Procedure Division contains instructions for solving a problem. The Procedure Division may contain imperative-statements, conditional statements, paragraphs, procedures, and sections (see "Procedure Division" for full details).

Procedure-name: A word that precedes and identifies a procedure, used by the programmer to transfer control from one point of the program to another.

Process: Any operation or combination of operations on data.

Program-name: A word in the Identification Division that identifies a COBOL source program.

Punctuation Character: A comma, semicolon, period, quotation mark, left or right parenthesis, or a space.

Qualifier: A group data-name that is used to reference a non-unique data-name at a lower level in the same hierarchy, or a section-name that is used to reference a non-unique paragraph. In this way, the data-name or the paragraph-name can be made unique.

Random Access: An access mode in which specific logical records are obtained from, or placed into, a mass storage file in a nonsequential manner.

RECORD: A set of one or more related data items grouped for handling either internally or by the input/output systems (see "Logical Record").

Record Description: The total set of data description entries associated with a particular logical record.

Record-name: A data-name that identifies a logical record.

Reel: A module of external storage associated with a tape device.

Relation Character: A character that expresses a relationship between two operands. The following are COBOL relation characters:

Character	Meaning
>	Greater than
<	Less than
=	Equal to

Relation Condition: A statement that the value of an arithmetic expression or data item has a specific relationship to another arithmetic expression or data item. The statement may be true or false.

Relational Operator: A reserved word, or a group of reserved words, or a group of reserved words and relation characters. A relational operator plus programmer-defined operands make up a relational expression. A complete listing is given in "Procedure Division."

REPORT: A presentation of a set of processed data described in a Report File.

Report Description Entry: An entry in the Report Section of the Data Division that names and describes the format of a report to be produced.

Report File: A collection of records, produced by the Report Writer, that can be used to print a report in the desired format.

REPORT FOOTING: A report group that occurs, and is printed, only at the end of a report.

Report Group: A set of related data that makes up a logical entity in a report.

REPORT HEADING: A report group that occurs, and is printed, only at the beginning of a report.

Report Line: One row of printed characters in a report.

Report-name: A data-name that identifies a report.

REPORT SECTION: A section of the Data Division that contains one or more Report Description entries.

Reserved Word: A word used in a COBOL source program for syntactical purposes. It must not appear in a program as a user-defined operand.

Routine: A set of statements in a program that causes the computer to perform an operation or series of related operations.

Run Unit: A set of one or more object programs that function, at object time, as a unit to provide problem solutions. This compiler considers a run unit to be the highest level calling program plus all called subprograms.

S-mode Records: Records that span physical blocks. Records may be fixed or variable in length. Blocks may contain one or more segments. A segment may contain one record or a portion of a record. Each segment contains a segment-length field and a control field indicating whether or not it is the first and/or last or an intermediate segment of the record. Each block contains a block-length field.

SECTION: A logically related sequence of one or more paragraphs. A section must always be named.

Section Header: A combination of words that precedes and identifies each section in the Environment, Data, and Procedure Divisions.

Section-name: A word specified by the programmer that precedes and identifies a section in the Procedure Division.

Sentence: A sequence of one or more statements, the last ending with a period followed by a space.

Separator: An optional word or character that improves readability.

Sequential Access: An access mode in which logical records are obtained from, or placed into, a file in such a way that each successive access to the file refers to the next subsequent logical record in the file. The order of the records is established by the programmer when creating the file.

Sequential Processing: The processing of logical records in the order in which records are accessed.

Sign Condition: A statement that the algebraic value of a data item is less than, equal to, or greater than zero. It may be true or false.

Simple Condition: An expression that can have two values, and causes the object program to select between alternate paths of control, depending on the value found. The expression can be either true or false.

Slack Bytes: Bytes inserted between data items or records to ensure correct alignment of some numeric items. Slack bytes contain no meaningful data. In some cases, they are inserted by the compiler; in others, it is the responsibility of the programmer to insert them. The SYNCHRONIZED clause instructs the compiler to insert slack bytes when they are needed for proper alignment. Slack bytes between records are inserted by the programmer.

Sort File: A collection of records that is sorted by a SORT statement. The sort file is created and used only while the sort function is operative.

Sort File Description Entry: An entry in the File Section of the Data Division that names and describes a collection of records that is used in a SORT statement.

Sort-file-name: A data-name that identifies a Sort File.

Sort-key: The field within a record on which a file is sorted.

Sort-work-file: A collection of records involved in the sorting operation as this collection exists on intermediate device(s).

SOURCE-COMPUTER: The name of an Environment Division paragraph. In it, the computer upon which the source program will be compiled is described.

Source Program: A problem-solving program written in COBOL.

Special Character: A character that is neither numeric nor alphabetic. Special characters in COBOL include the space ( ), the period(.), as well as the following:

$$+ \quad - \quad * \quad / \quad = \quad \$ \quad , \quad ; \quad " \quad) \quad ($$

SPECIAL-NAMES: The name of an Environment Division paragraph, and the paragraph itself, in which names supplied by IBM are related to mnemonic-names specified by the programmer. In addition, this paragraph can be used to exchange the functions of the comma and the period, or to specify a substitution character for the currency sign, in the PICTURE string.

Special Register: Compiler-generated storage areas primarily used to store information produced with the use of specific COBOL features. The special registers are: TALLY, LINE-COUNTER, PAGE-COUNTER, CURRENT-DATE, TIME-OF-DAY, COM-REG, SORT-RETURN, SORT-FILE-SIZE, SORT-CORE-SIZE, SORT-MODE-SIZE, and NSTD-REELS.

Standard Data Format: The concept of actual physical or logical record size in storage. The length in the Standard Data Format is expressed in the number of bytes a record occupies and not necessarily the number of characters, since some characters take up one full byte of storage and others take up less.

Statement: A syntactically valid combination of words and symbols written in the Procedure Division. A statement combines COBOL reserved words and programmer-defined operands.

Subject of entry: A data-name or reserved word that appears immediately after a level indicator or level number in a Data Division entry. It serves to reference the entry.

Subprogram: A COBOL program that is invoked by another COBOL program. (Programs written in other languages that follow COBOL linkage conventions are COBOL programs in this sense.)

Subscript: An integer or a variable whose value references a particular element in a table.

Switch-status Condition: A statement that an UPSI switch has been set to an ON or OFF condition. The statement may be true or false.

SYSIPT: The system input device.

SYSLST: The system output device.

SYSPCH: The system punch device.

SYSPUNCH: An alternate name for the system punch device.

System-name: A name, specified by IBM, that identifies any particular external device used with the computer, and characteristics of files contained within it.

Table: A collection and arrangement of data in a fixed form for ready reference. Such a collection follows some logical order, expressing particular values (functions) corresponding to other values (arguments) by which they are referenced.

Table Element: A data item that belongs to the set of repeated items comprising a table. An argument together with its corresponding function(s) makes up a table element.

Test Condition: A statement that, taken as a whole, may be either true or false, depending on the circumstances existing at the time the expression is evaluated.

Trailer Label: A record that identifies the ending of a physical file or of a volume.

U-mode Records: Records of undefined length. They may be fixed or variable in length; there is only one record per block.

Unary Operator: An arithmetic operator (+ or -) that can precede a single variable, a literal, or a left parenthesis in an arithmetic expression. The plus sign multiplies the value by +1; the minus sign multiplies the value by -1.

UNIT: A module of external storage. Its dimensions are determined by IBM.

V-mode Records: Records of variable length, each of which is wholly contained within a block. Blocks may contain more than one record. Each record contains a record length field, and each block contains a block length field.

Variable: A data item whose value may be changed during execution of the object program.

Verb: A COBOL reserved word that expresses an action to be taken by a COBOL compiler or an object program.

<u>Volume</u>: A module of external storage. For tape devices it is a reel; for mass storage devices it is a unit.

<u>Volume Switch Procedures</u>: Standard procedures executed automatically when the end of a unit or reel has been reached before end-of-file has been reached.

<u>Word</u>:

1. In COBOL: A string of not more than 30 characters, chosen from the following: the letters A through Z, the digits 0 through 9, and the hyphen (-). The hyphen may not appear as either the first or last character.

2. In System/360: A fullword is 4 bytes of storage; a doubleword is 8 bytes of storage; a halfword is 2 bytes of storage.

<u>Word Boundary</u>: Any particular storage position at which data must be aligned for certain processing operations in System/360. The halfword boundary must be divisible by 2, the fullword boundary must be divisible by 4, the doubleword boundary must be divisible by 8.

<u>WORKING-STORAGE SECTION</u>: A section-name (and the section itself) in the Data Division. The section describes records and noncontiguous data items that are not part of external files, but are developed and processed internally. It also defines data items whose values are assigned in the source program.
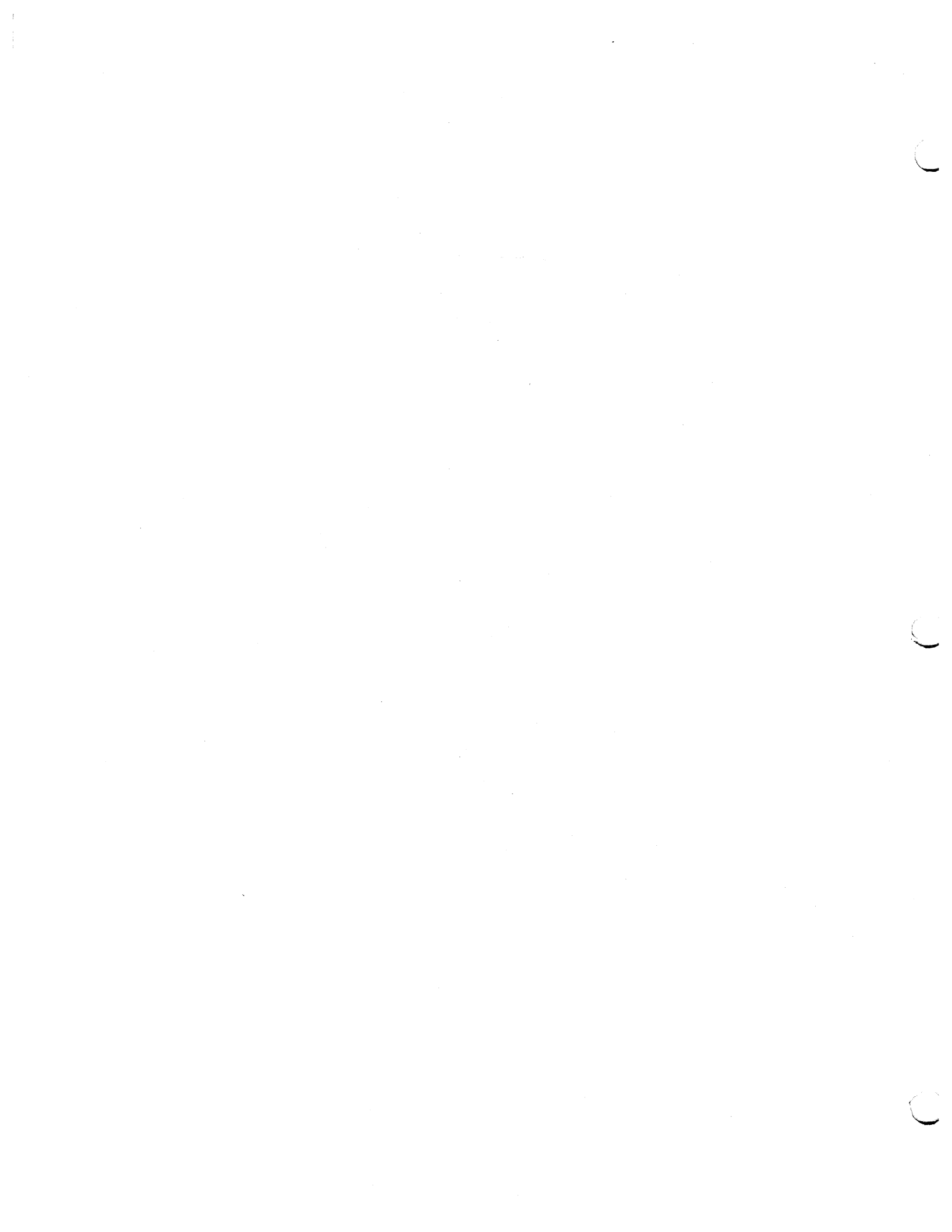
(When more than one page reference is given, the major reference appears first.)

Special Characters

[(see braces)
{ (see brackets)
  (see pound sign)
. (see period)
... (see ellipsis)
< used in relation conditions   142,143
(and) used in
  arithmetic expressions   137,138
  compound conditions   147-149
  PICTURE clause   113,114
  subscripting and indexing   269-271
+ (see plus symbol)
$ (see currency symbol, dollar sign)
* used in arithmetic expressions
 137,138,311
  (see also asterisks, used in PICTURE
  clause)
** used in arithmetic
 expressions   137,138,311
; used in Data Division and Procedure
 Division entries   36,100
  (see also semicolon)
- (see either hyphen, or minus symbol)
/ used in
  arithmetic expressions   137,138,311
  sterling report items   306-308
, (see comma)
> used in relation conditions   142,143
= used in
  COMPUTE statement   163
  relation conditions   142,143
' or " used in nonnumeric literals   40,35
  (see also quotation mark)

A, used in a PICTURE clause   108,112,114
  alphabetic items   112
  alphanumeric edited items   114
  alphanumeric items   112
abbreviations
  in CORRESPONDING option   160,162,166
  in END-OF-PAGE option   193
  in Identification Division Header   55
  in JUSTIFIED clause   106
  in PICTURE clause   107
  of relation conditions   148,149
  of relational operators   142
  of reserved words   38
  in SYNCHRONIZED clause   118.2
  in TYPE clause   247
  in USAGE clause   124
absolute
  column number   249
  LINE clause   244,245
  line spacing in a report   244,245
  values in MOVE statement   180

ACCEPT statement   197,198
access, definition   335
access methods
  for direct files   58-60
  for indexed files   59,60
  for sequential files   58,59
ACCESS MODE clause   71,72,70
actual decimal point
  description   109
  in special insertion editing   115
  in zero suppression editing   118
ACTUAL KEY clause
  description   72-74
  with direct files
    creation   59
    general use   58,59
    random access   59
    sequential access   58
  examples   74,31,32,314,315
  format   72
  and READ statement   191,192
  record identifier component   74,73
  and REWRITE statement   197
  and SEEK statement   190
  symbolic portion   74,73
  track identifier components   72-74.1
  and WRITE statement   192,196
ADD statement
  CORRESPONDING option   162,160
  description   162,163
  examples   15,22,315
  formats   162
  GIVING option   162
  SIZE ERROR option   162,163
addition   162,163
addition operator   137,138
addressing schemes
  direct   58,59,72,73
  indexed   59,75,76
  sequential   58
  track identifier   59,72-74.1
ADVANCING option of the WRITE statement
 193,194
AFTER ADVANCING option of the WRITE
 statement   193-195,232
AFTER POSITIONING option of the WRITE
 statement   193-195
algebraic value in a sign condition   146
algorithm
  ACTUAL KEY example   314,315
  direct indexing   276,277
  for intermediate results   311,312
  relative indexing   279
  slack bytes
    and blocked records   123
    elementary computational items   119
    group items with an OCCURS clause
     120-123

compiler options
    current-date   42
    quotation mark   35
    sequence checking   47
    truncation   108
compound conditions
    definition   336
    evaluation rules   147
    implied subjects and
      relational-operators   148,149
    logical operators   146
    permissible symbol pairs   148
    SEARCH statement   283
COMPUTATIONAL items (see binary data items)
COMPUTATIONAL option of the USAGE clause
    124-124.5,121,122
COMPUTATIONAL-1 items (see short precision
  internal floating-point items)
COMPUTATIONAL-1 option of the USAGE clause
    124,124.2,121,122
COMPUTATIONAL-2 items (see long precision
  internal floating-point data items)
COMPUTATIONAL-2 option of the USAGE clause
    124,124.2,121,122
COMPUTATIONAL-3 items (see internal decimal
  items)
COMPUTATIONAL-3 option of the USAGE clause
    124,124.2,121,122
COMPUTE statement
    description   163
    example   22,32
    format   163
    ROUNDED option   160,161
    SIZE ERROR option   161
computer-name
    OBJECT-COMPUTER paragraph   63
    SOURCE-COMPUTER paragraph   62
COM-REG special register   42
condition, definition   336,139
conditional sentence, definition   150
conditional statement
    in debugging   300,300.1
    definition   336,150
    examples   23,24,26,28,32
    IF statement   150-152
    list of   135
    ON statement   300
conditional variable
    assigning values to   125,127,100,141
    condition-name condition   141
    definition   336
    example   100,127,141
    qualification of   45
condition-name
    assigning values to   125,127,100,141
    definition   336
    example   100,127,141
    format   100,141
    qualification   45
    range of values   125,127
    REDEFINES clause   103
    in switch-status condition   146,64,65
    VALUE clause   125,127,100,103
condition-name condition
    definition   337
    description   141
    format   141
    in a SEARCH statement   283

condition-name entries, REDEFINES clause
  103
conditions
    compound conditions   146-149
    in a PERFORM statement   170,172-176
    in a SEARCH statement   282-286
    test conditions   139-146
Configuration Section
    copying   293
    description   62
    example   31,287
    format   62
    OBJECT-COMPUTER paragraph   62,63
    SOURCE-COMPUTER paragraph   62
    SPECIAL-NAMES paragraph   63-65
connective, definition   337
CONSOLE
    in ACCEPT statement   197,198
    definition   337
    in DISPLAY statement   199
    in SPECIAL-NAMES paragraph   64
constant
    definition   39
    figurative   40,41
    literal   39,40
contiguous items, definition   337
continuation area
    in comment lines   49,213
    in reference format   47
continuation line   47,48
continuation of
    comments   49,213
    lines   47,48
    nonnumeric literals   48
    numeric literals   48
    words   48
continued line   50,51
control breaks   237,238,232,233,253,254
control bytes
    BLOCK CONTAINS clause   93
    definition   337
    and inter-record slack bytes   123
    in S-mode records   95
    in V-mode records   95
CONTROL clause   237,238
control data item, definition   337
CONTROL FOOTING report group
    CONTROL clause   237,238
    definition   337
    incrementing counters   252,253
    LINE clause   245,246
    NEXT GROUP clause   246
    PAGE LIMIT clause   239,240
    report group description entry   241
    summation   252
    TERMINATE statement   255
    TYPE clause   247,248
control group, definition   337
CONTROL HEADING report group
    CONTROL clause   237,238
    definition   337
    GENERATE statement   254
    LINE clause   245
    NEXT GROUP clause   245
    PAGE LIMIT clause   238
    report group description entry   241
    TYPE clause   247-249
control hierarchy   237,238

external decimal items   124,124.1
external floating-point items
  124,124.1
numeric edited items   124
SYNCHRONIZED clause   118.3
DISPLAY-ST data item   303-307
DISPLAY-ST option of the USAGE clause
  304-308,306
DISPLAY statement   198-200
DIVIDE statement
    description   164
    formats   164
    GIVING option   164,160
    REMAINDER option   164
    ROUNDED option   164,160,161
    SIZE ERROR option   164,161
dividend   164
division, arithmetic operation   164
division by zero   164,161
division header
    definition   338
    description   48
division-name,definition   338
division operator   137,138,37
division of a program, definition   338
dollar sign (see currency symbol, dollar
  sign)
double spacing
    printer page   194,195
    source program listing   302
doubleword
    binary items   124.2
    SYNCHRONIZED clause   118.3
DOWN BY option of the SET statement   286
dummy files   67,188,192,203
duplication of names in a source program
  45,46

E, used in
    external floating-point items
      113,124.1
    floating-point numeric literals   40
EBCDIC character, definition   338
EBCDIC collating sequence (Extended Binary
  Coded Decimal Interchange Code)   143,224
editing
    insertion
        fixed   116
        floating   116,117
        simple   115
        special   115
    replacement   117,118
    sign control symbols
        description   108-110
        in fixed insertion editing   116
        in floating insertion editing
          116,117
        in sterling report items   308
    symbols
        in alphanumeric edited items   114
        in arithmetic statements   160
        description   108-110
        in numeric edited items   114
        in SUM counter description   252
    zero suppression   117,118

editing character
    definition   338
    description   108-110,37
    insertion
        fixed   116
        floating   116,117
        simple   115
        special   115
EJECT statement   302
elementary item
    definition   339
    description   87,88
    (see also data description clauses)
    renaming   128-130,88
    slack bytes   119,120
    SYNCHRONIZED clause   118.2-119
ellipsis (...) in formats   51
ELSE option
    IF statement   150-152
    nested IF statements   151,152
    ON statement   300.1
END DECLARATIVES.   153,133,134
end of file
    when reading   191,192
    when sorting   228
end of page condition   195
END-OF-PAGE (EOP) option of the WRITE
  statement   195
end of volume positioning   201-203,188
ENTER statement   212
entry, definition   339
entry-name, definition   339
entry point   205-207
ENTRY statement   206,207
Environment Division
    ASCII considerations   334.1,334.2
    Configuration Section
        definition   339
        description   62-65
        OBJECT-COMPUTER paragraph   63
        SOURCE-COMPUTER paragraph   62
        SPECIAL-NAMES paragraph   63-65
    Input-Output Section
        definition   340
        FILE-CONTROL paragraph   66-76
        I-O CONTROL paragraph   77-82
    international considerations   303
    organization   61
    segmentation considerations   291,62,63
    sort considerations   217-221
EOP option of the WRITE statement   195
equal sign (=)
    in COMPUTE statement   163
    in relation condition   142
equal size operands in a relation
    condition   144
equivalents
    reserved words and abbreviations   38
    THROUGH and THRU   38
error bytes   157,158
error conditions, arithmetic operations
  (see SIZE ERROR option in arithmetic
  statements)
ERROR PROCEDURE option of USE statement
  156.4-158
error processing declaratives
    and ACTUAL KEY   74
    description   156.4-158

and RESET clause 250-251
resetting to zero 250,251,254,255
summary reporting 253
summation in a report 250-255
SUPPRESS option of COPY statement 294
suppress spacing 194,195
suppression of
 library entry listing 294
 printing of a report group 256,257
 sequence checking 47
 spacing in WRITE statement 194,195
 zeros in PICTURE clause 109,117,118
suppression and replacement editing
117,118
suppression symbols 109,117
switch-status condition
 condition-name in 146
 definition 146
 and system-names 146,64,65
symbol pair in a compound condition 148
symbolic debugging, Version 3 334.8
symbolic portion of ACTUAL KEY 74
symbols
 in arithmetic expressions
  137,138,37
 in floating-point literals 40
 in PICTURE clause 108-110
 in relation conditions 142,37
 in sterling currency formats 303
SYNCHRONIZED clause
 description 118.2-123
 format 118.2
 index data items 279,280
 and OCCURS clause 120-122
 slack bytes 119-123
SYSIPT 197,198,64
SYSLST 199,64
SYSnnn field in system-name 68
SYSPCH and SYSPUNCH 199,64
system closing conventions 201-204
system features
 COM-REG special register 42
 CURRENT-DATE special register 42
 NSTD-REELS special register 42,68
 TALLY special register
  41,182,183,251,252,269
 TIME-OF-DAY special register 42
 UPSI (user program status indicator)
  switches 64,65
system logical input device 197,198,64
system logical output device 198,199,64
system-name
 in ASSIGN clause 68,69
 definition 346
 in RERUN clause 77,78,220
 in the Sort Feature 220
system procedures (see standard system
 procedures)
system routines
 error 156.4
 label handling 155
system switches 64,65
System/370 support, Version 3 63,334.8
SYS001,SYS002 in Sort 219
S01 and S02 function-names
 definition 64

table, definition 346
table elements 273,269,346
Table Handling
 ascending/descending sequence 275,278
 Data Division considerations
  OCCURS clause 272-279
  USAGE clause 279
 examples 271,272,276-278,285,287,288
 indexing
  description 276-278,270,271
  direct 278,270
  relative 278,279,270
 Procedure Division considerations
  relation conditions 280,281
  SEARCH statement 281-285
  SET statement 286
 sample program 287,288
 subscripting 269-271
TALLY special register
 in ACCEPT statement 198
 description 41
 in DISPLAY statement 199
 in the EXAMINE statement 182,183
 in a SOURCE clause 251
 as a subscript 269
 in a SUM clause 252
tape device, error information 157,158
tape file, label handling
 description 97,154-156.3
 and NO REWIND option 188,201-203
 and REVERSED option 188,201-203
 and USE statement 154-156.3
TERMINATE statement 255,256,233
termination of
 execution 177
 main programs 210,211,177
 report processing 255,256
 subprograms 210,211
test conditions
 class 140
 compound 146-149
 condition-name 141
 definition 346
 relation 142-145
 sign 146
 switch-status 146
THEN
 used in IF statement 150
 used in sentences 36,133
THRU option
 of the PERFORM statement 170,172-176
 and THROUGH, equivalence of 38
 of the VALUE clause 125,127
TIME-OF-DAY special register description
42
TIMES option of the PERFORM statement
170-172
TO operand of the TRANSFORM statement
184-186
TRACE statement 298
track address
 algorithm example 314
 component of ACTUAL KEY 72-74.1
 direct file 57,72-74,74.1,314
TRACK-AREA clause 76
trailer labels 154-156.3,97

GC28-6394-2

IBM S/360 DOS Full Amer Natl Std COBOL   Printed in U.S.A.   GC28-6394-2

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

IBM / **Technical Newsletter**

IBM SYSTEM/360 DISK OPERATING SYSTEM
FULL AMERICAN NATIONAL STANDARD COBOL
© IBM Corp. 1968, 1969, 1970, 1971

This Technical Newsletter provides replacement pages for IBM System/360 Disk Operating System: Full American National Standard COBOL, Order No. GC28-6394-2. For the Version 1 and Version 2 Compiler, it corresponds ████████████████████████████████ Changes for the Program ████████████ No. 5736-CB2 and 5736-LM2, are also included. Pages to be inserted are:

Cover-2	107-118.3	223-224
3-4	119-120	261-262
7-14.1	123-124.5	273-276.1
29-30	125-126	293-294
39-40.1	145-146	297-300.1
41-42	153-156.4	305-308.1
61-70.1	161-162	309-312
73-74.1	171-172	321-328
77-78	193-196.1	334.1-334.9
93-98.1	197-198	335-336
103-104	217-218.1	339-342.1
		Index

Summary of Amendments

Descriptions of Program Product Version 3 items: the addition of the SIGN clause, and ASCII capabilities, and a list of Version 3 features, as well as changes in the implementation of the OBJECT-COMPUTER paragraph and the ON statement.

Changes for all Versions include: clarifications of LABEL RECORDS, PICTURE and USAGE clause descriptions, addition of a PICTURE table of precedence, and a sample program for user label processing. Many examples have been added.

Minor omissions and errors that appeared in the previous edition have also been corrected.

A change to the text or a small change to an illustration is indicated by a vertical line to the left of the change. A changed or added illustration is denoted by the symbol • to the left of the caption.

Note: Please file this cover letter at the back of the manual to provide a record of changes.

PRINTED IN U.S.A.

4/19/72

IBM DOS FULL AMERICAN
NATIONAL STANDARD COBOL
© IBM Corp. 1968, 1969, 1970, 1971

This Technical Newsletter provides new pages and replacement pages
for IBM DOS Full American National Standard COBOL, Order No. GC28-
6394-2 as amended by TNL GN28-0436. Changes for Release 2 of the
Version 3 Program Product Compiler, Program Nos. 5736-CB2,
5736-CB3, and 5736-LM2, are included. Pages to be inserted are:

Front Cover	81-82
Preface	291-292
Table of Contents	309
11-14	323-324
69-70.4	334.7-334.28
77-78.1	

Summary of Amendments

Documentation for Version 3 Release 2 System/370 device support is
added. Two new appendixes explain the Version 3 symbolic debug
feature and Version 3 3525 combined function processing.

Changes to text are indicated by a vertical line to the left of the
change. Added illustrations have the symbol • to the left of the
caption.

Note: Please file this cover letter at the back of the manual
to provide a record of changes.