HONEYWELL EDP

SOFTWARE MANUAL

SERIES 200

COBOL COMPILER B

GENERAL SYSTEM:	SERIES 200/BASIC PROGRAMMING SYSTEM
SUBJECT:	Programming and Operating Procedures for COBOL Compiler B.
SPECIAL INSTRUCTIONS:	This manual completely supersedes the preliminary software bulletin COBOL Compiler B (File Number 122.1205.000B.1-028) dated January 14, 1966.

DATE: April 25, 1966

FILE NO:. 123.1205.000B.0-292*

8828
7366
Printed in U.S.A.

*When ordering this publication please specify Title and Underscored portion of File Number.

PREFACE

This software manual is intended to be used only for reference and recall purposes by programmers having previous COBOL knowledge or experience. It is not intended to be a self-teaching guide. Those with no previous COBOL background are directed to the publication set entitled Study Guide: COBOL Programming, Order Number 260. This study guide, consisting of three books, begins with the basic elements of computers and data processing in general, continues through an explanation and examples of each basic language element, and concludes with a series of sample programming techniques and routines. A companion publication entitled Study Guide: Easytab System, Order Number 281, is also suggested as a prerequisite for those using COBOL B as part of the Easytab programming system.

This reference manual is organized according to the logical path followed in writing a COBOL program. The order also parallels that of the reference manuals for the other Honeywell COBOL compilers to facilitate cross-referencing among them.

The reader has a choice of using this manual in its bound form (as distributed) or inserting it in a three-ring binder with the staples removed. The loose-leaf usage permits easy updating by means of addenda, which are distributed in the same manner as the manual. The cover of each addendum indicates the pages to be replaced and the reasons for the revisions. Revision pages are dated and indicate, via bars in the margin, what changes have been made in the text. The only exception to this notation occurs when an entire section is replaced. In this case, the change or addition is noted on the cover and no bars appear in the margin. When a revision page replaces an existing page, the existing page is discarded and the new page inserted. When a page is to be inserted between existing pages, a decimal page number is used. For example, a page to be inserted between pages 4 and 5 of Section VI is numbered 6-4.1. Whenever a revision is made to the manual, it is suggested that the user enter the date of the revision on the log sheet provided in the front. In this way a record can be kept which will aid in ascertaining that all available revisions have been received and entered in the manual.

ACKNOWLEDGEMENT

The following acknowledgement has been reproduced from COBOL Report to Conference on Data Systems Languages, U. S. Department of Defense, 1961, at the request of the Conference.

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

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

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

Allstate Insurance Company
Bendix Corporation, Computer Division
Control Data Corporation
DuPont Corporation
General Electric Company
General Motors Corporation
Lockheed Aircraft Corporation
National Cash Register Company
Philco Corporation
Standard Oil Company (N. J.)
United States Steel Corporation

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

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

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

The authors and copyright holders of the copyrighted material used herein: FLOW-MATIC (Trademark of Sperry-Rand Corporation), Programming for the UNIVAC® I and II, Data Automation Systems © 1958, 1959, Sperry-Rand Corporation; IBM Commercial Translator, Form No. F 28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Honeywell Inc. have specifically authorized the use of this material, in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications.

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

MANUAL REVISIONS

As each set of revision pages is received and inserted into this manual, enter the revision date below.

/ /	/ /	/ /	/ /
/ /	/ /	/ /	/ /
/ /	/ /	/ /	/ /
/ /	/ /	/ /	/ /
/ /	/ /	/ /	/ /
/ /	/ /	/ /	/ /
/ /	/ /	/ /	/ /
/ /	/ /	/ /	/ /
/ /	/ /	/ /	/ /
/ /	/ /	/ /	/ /

TABLE OF CONTENTS

TABLE OF CONTENTS (cont)

LIST OF ILLUSTRATIONS

LIST OF TABLES

INTRODUCTION

This publication defines the COBOL language elements available for the Honeywell Series 200 COBOL Compiler B, which operates on any Series 200 computer having a minimum of 8,192 characters of main memory and the other equipment outlined in Table 0-1. The compiler operating procedures and input/output considerations are also presented, along with a discussion of compiler listings and diagnostic messages. Descriptions of both the Loader B and Update B programs are included in Appendix C and Appendix D.

COBOL is an easy-to-learn programming language designed primarily for commercial applications. Using a subset of the English language, COBOL follows format and usage conventions which are familiar to the English-speaking person. Among the major advantages of COBOL are: simplicity - programs coded in COBOL language can be written, read, and understood by non-technical personnel with minimal background in the data processing field; shorter training time - training time is reduced, not only for the novice but also for the experienced COBOL programmer being retrained for another computer; compatibility - programs written for one model of computer can easily be modified to run on other models, either of the same or of another manufacturer. When used as part of the Easytab[1] programming system, COBOL B is particularly useful in programming tab operations not covered by any of the precoded Easytab Utility Routines.

EQUIPMENT REQUIREMENTS

The Series 200 equipment which is required or which may be used with COBOL Compiler B is listed in Table 0-1.

Table 0-1. Equipment Requirements for COBOL Compiler B

EQUIPMENT	AT COMPILE TIME		AT OBJECT TIME	
	Quantity Required	Additional Usable	Quantity Required	Additional Usable
Central Processor with 8K characters of main memory	1	0	1	0
Additional memory up to a total of 32K characters	0	1	0	1

[1] The Easytab programming system is further described in the software announcement: Easytab, Order Number 104. Programming and operating information for the Easytab Utility Routines can be found in the software manual entitled Easytab Utility Programs, Order Number 206.

Table 0-1 (Cont). Equipment Requirements for COBOL Compiler B

EQUIPMENT	AT COMPILE TIME		AT OBJECT TIME	
	Quantity Required	Additional Usable	Quantity Required	Additional Usable
Advanced Programming Instructions	1	0	1	0
Editing Instructions	0	0	1	0
Tape Control (1/2-inch tapes)	1	0	0	1
Magnetic Tape Units (1/2-inch tapes)	2	1[1]	0	8
High-Speed Printer	1	0	0	1
Extension of Print Line to 132 Characters	0	0	0	1
Card Reader/Punch	1[1]	0	0	1

[1]The additional output device required may be either another tape unit or a card punch.

SYMBOLOGY

The symbology adopted for the representation of values in examples and illustrations is that used in the Department of Defense publication Report to Conference on Data Systems Languages, 1961. The CODASYL notation is as follows:

1. LOWER-CASE CHARACTERS represent information that must be supplied by the programmer.
2. UPPER-CASE CHARACTERS THAT ARE UNDERLINED are key words and must be used when the functions of which they are a part are used.
3. UPPER-CASE CHARACTERS THAT ARE NOT UNDERLINED are words reserved by the COBOL compiler but that are optional when the functions of which they are a part are used. They are generally used as documentation.
4. BRACES $\left\{ \quad \right\}$ indicate that a choice must be made from the contents enclosed by the braces.
5. SQUARE BRACKETS $\left[\quad \right]$ indicate that the contents enclosed are optional and can be included in the source program or omitted, as desired.
6. SERIES NOTATION. If two or more nouns can be written in a series in a COBOL statement, commas are shown as connectives in the format specification. Where a comma is shown as a connective in a format specification, the comma can be omitted or it can be replaced by "AND" or by ",ΔAND."

SECTION I

ELEMENTS OF COBOL B LANGUAGE

PROGRAM STRUCTURE

Honeywell COBOL programs consist of four major divisions:

IDENTIFICATION DIVISION - which consists of an indentification of the source program.

ENVIRONMENT DIVISION - which describes the equipment configuration on which the object program is to be compiled, the configuration of the equipment on which the object program is to run, and the relationship between data files and input/output media.

DATA DIVISION - which describes the data to be processed by the object program.

PROCEDURE DIVISION - which describes the procedures used in manipulating the data.

CHARACTER SET

The complete character set for Honeywell COBOL B consists of the following 45 characters

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, (numeric characters)

A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z (alphabetic characters)

blank or space (represented by a Δ)

- (hyphen or minus sign)

* (asterisk)

$ (dollar sign)

, (comma)

. (period or decimal point)

" (quotation mark)

((left parenthesis)

) (right parenthesis)

Of the above set, the following 37 characters are used for COBOL B words (which can be from one through thirty characters in length):

A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

- (hyphen or minus sign)

The characters used for punctuation are:

"	(quotation mark)
Δ	(blank or space)
(	(left parenthesis)
)	(right parenthesis)
,	(comma)
.	(period)

Additional characters, used in editing are:

$	(dollar sign)
*	(check protection symbol)
,	(comma)
.	(actual decimal point)

All of the above characters are contained in the COBOL character set. In addition, the Series 200 COBOL programmer can use as characters in non-numeric literals (see below) or in the sequence number field any characters included in the Series 200 high-speed printer character set that are not included in the COBOL set or any character that can be punched on a card; however, such characters, acceptable to all Honeywell COBOL compilers, may be unacceptable to COBOL compilers for other computers.

WORDS

A word is composed of not more than 30 characters chosen from the 37 characters given in the second character set above. The first 6 characters of a word must be unique within the source program. A word must contain at least one alphabetic character.

A word is ended by a space or by a period, right parenthesis, or comma. When a word is ended by a punctuation mark, the punctuation mark must be followed by a space. While a word can contain one or more hyphens, a hyphen cannot be the first or the last character of the word. A literal is a special type of word that is an exception to these rules and is discussed below.

PUNCTUATION

When a period or a comma, is used, it must immediately follow a word (no space can intervene) and it must be followed by a space. A left parenthesis must not be followed immediately by a space, and a right parenthesis must not be immediately preceded by a space. A beginning quotation mark must not be succeeded by a space, and an end quotation mark must not be preceded by a space unless such spaces are desired in the literal (see page 1-3).

LITERALS

A literal is an item of data, the value of which is identical to those characters constituting the literal. It can belong to one of two classes: non-numeric or numeric.

Non-Numeric Literals

Non-numeric literals must be bounded by quotation marks. These must immediately precede the first character of the literal and immediately succeed the last character.

A non-numeric literal can be composed of any allowable characters (see "Character Set," page 1-1) except the quotation mark. All spaces which are enclosed within the quotation marks bounding a literal are included as spaces in the literal.

A non-numeric literal can be from 1 through 30 characters in length.

Numeric Literals

A numeric literal must contain at least one numeric character. Under no conditions can it contain an alphabetic character.

In addition, a numeric literal can also contain as its leftmost character a plus (+) or a minus (-) sign. If neither sign is used, the literal is assumed to be positive.

A decimal point (.) can appear within the numeric literal except to the left of a + or a - sign or as the rightmost character of the literal. Such a point is treated as an implied decimal point. If the decimal point is not used, the literal is assumed to be an integer.

A numeric literal can be from 1 through 30 characters in length. However, it should not exceed 18 characters in length if compatibility with other manufacturers' COBOL compilers is a concern.

A literal conforming to the rules for the formation of numeric literals but enclosed within quotation marks is considered to be a non-numeric literal and is treated as such by COBOL Compiler B. That is,

-125.65 is not the same as "-125.65".

FIGURATIVE CONSTANTS

Certain values have been assigned fixed data-names and are called figurative constants. When used as figurative constants, they must not be bounded by quotation marks. If these fixed data names are bounded by quotation marks, they are considered to be non-numeric literals. The fixed data-names and their meanings are:

ZERO	Represents one or more ocurrences of the value 0 or the character 0, depending on context.
SPACE	Represents one or more blanks or spaces.
QUOTE	Represents one or more occurrences of the quotation mark (") at object time. The data-name QUOTE cannot be used to bound a non-numeric literal.

Figurative constants generate a string of homogeneous information whose length is determined by the compiler based upon context. When the length is not deducible from context, a single character is generated. The figurative constants can be used only in the PROCEDURE DIVISION and the DATA DIVISION of the source program.

SOURCE-LANGUAGE RESERVED WORDS

The source language consists of reserved and non-reserved words. Reserved words are those which are required to express a procedure function (such as ADD or MOVE) or to describe a dat unit, or words which can be used to make a program easier to read.

Reserved words whose use is mandatory are called key words. Reserved words whose use is at the direction of the programmer are called optional words. A complete list of the reserved words for Series 200 COBOL is given in Appendix A.

SOURCE-LANGUAGE USER-CREATED NAMES (NON-RESERVED WORDS)

User-created names are non-reserved words created by the programmer and used as literals, data-names, or procedure-names. User-created names may have up to 30 characters. However, only the first six characters are recognized by the compiler and these six characters must form a unique name within the program environment. The programmer must be aware that the reserved words listed in Appendix A cannot serve as user-created names.

GENERAL SYNTACTICAL STRUCTURE OF COBOL SOURCE LANGUAGE

COBOL procedures are expressed in a manner similar to, though not identical to, normal English prose. The basic unit of procedure formation is a sentence or a group of successive sentences. A procedure is a paragraph or a group of successive paragraphs within the PROCEDURE DIVISION. The following discussion is more fully developed in Section VI.

Statements

A statement expresses a processing function or a condition. In general, a statement consists of a verb and its operand, or a condition together with its subjects and objects. There are three types of statements:

1. An imperative statement consists of either a verb (excluding compiler-directing verbs, see Section VI) and its operands or a sequence of imperative statements. A sequence of imperative statements can contain either a GO TO or a STOP RUN imperative statement. If either is present, it must appear as the last imperative statement of its (GO TO or STOP RUN) sequence.
2. A conditional statement takes the form:

   IF condition imperative-statement.
3. A compiler-directing statement consists of a compiler-directing verb and its operands.

Sentences

A sentence can be either imperative, conditional, or compiler-directing.

IMPERATIVE SENTENCES

An imperative sentence consists of at least one imperative statement, the last of which is terminated by a period. For example, both

```
ADD A TO B.
ADD A TO B  MOVE B TO C.
```

are imperative sentences.

CONDITIONAL SENTENCES

A conditional sentence consists of a conditional statement terminated by a period, such as

```
IF A EQUAL TO B GO TO C.
```

COMPILER-DIRECTING SENTENCES

A compiler-directing sentence consists of a compiler-directing statement terminated by a period. For example, both

```
EXIT.
NOTE THAT THE FOLLOWING PARAGRAPH CONSTITUTES THE
ENTIRE INPUT-OUTPUT SECTION OF THIS PROGRAM.
```

are compiler-directing sentences.

PUNCTUATION OF SENTENCES

The following rules apply to the punctuation of sentences.

1. A sentence is terminated by a period.
2. In the COBOL B environment, a comma is the only legal separator. Use of the comma is optional; when used, it must not be followed by another comma.

Paragraphs

A paragraph is a sentence or a group of sequential sentences to which a procedure-name (known as a paragraph-name) is assigned. Because paragraphs are named, they can be referenced from other parts of the PROCEDURE DIVISION of the source program.

Sentences are executed in the order of their appearance within a paragraph. This is subject to the results of any tests of conditions in the sentences (since such tests could result in control being transferred to another paragraph). Paragraphs are executed in their order of appearance within the program, barring any specified transfers of control.

SUBSCRIPTING

The technique of subscripting is most commonly used for table-handling functions. The ability to reference individual elements (of a table or list) which have not been assigned individual data-names is provided by using subscripts; the ability to reference the entire table or list is provided by using the name of the table or list.

A subscript is an integer whose value determines which element is being referred to within a table (or list) of like elements. The subscript may be represented either by a literal which is an integer (e. g. , 25) or by a data-name (e. g. , AGE) which has an integral value.

When the subscript is represented by a data-name, the data-name must be described by a record description entry in the DATA DIVISION. In both cases, i. e. , whether the subscript is represented by a literal or a data-name, the subscript is enclosed in parentheses and appears immediately after the terminal space of the name of the element referenced, e. g. , RATE (AGE) or RATE (25). One level of subscripting is permitted.

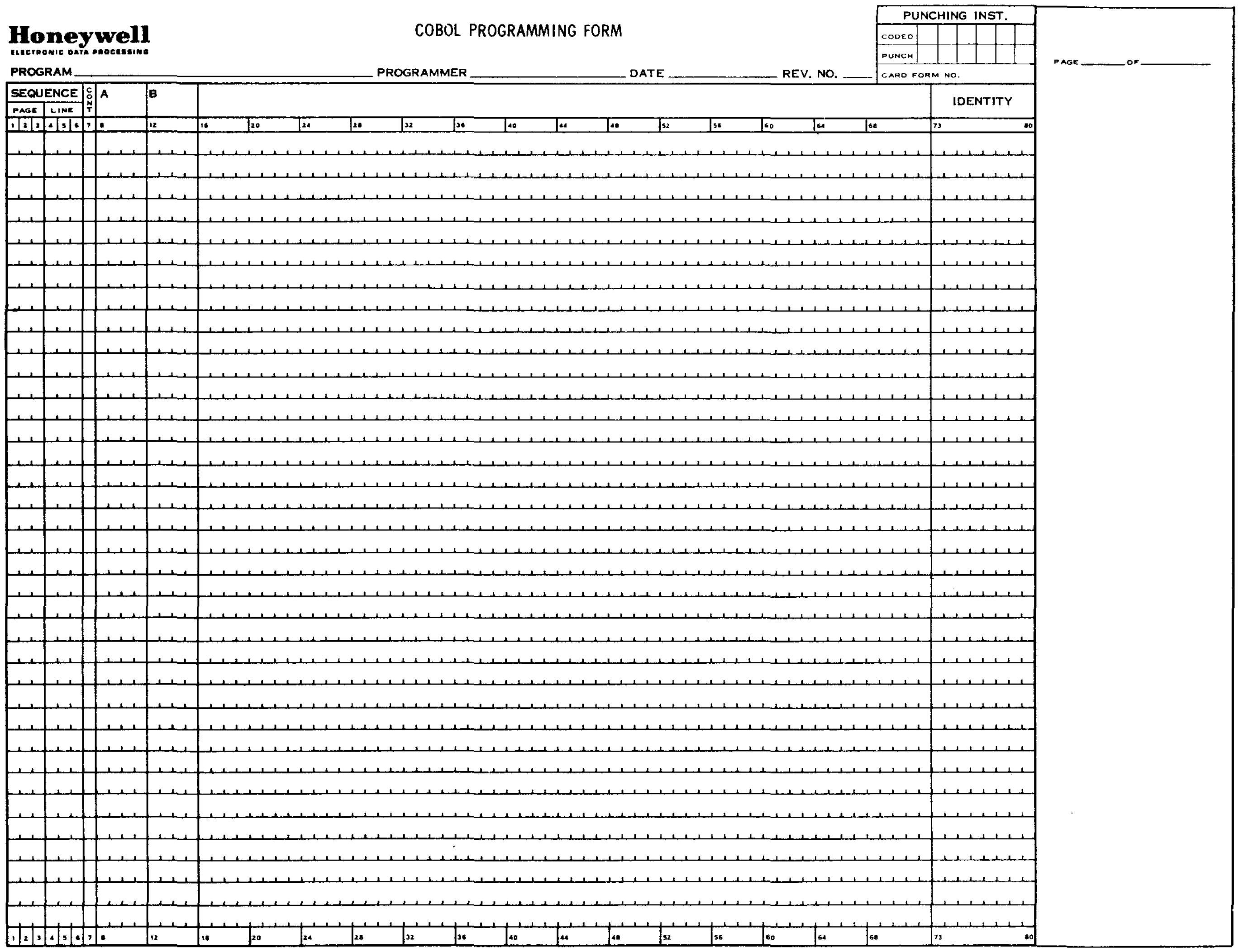
No element of a table or list can be referenced without a subscript. However, the entire table can be referenced, provided that the table has been given a unique name. Some examples of the writing of subscripts are:

```
MOVE RATE (AGE) TO LISTING.
IF HEIGHT (10) IS GREATER THAN......
MULTIPLY PRICE (STOCK-NO) BY INVENTORY (STOCK-NO).
EXAMINE STATE (REGION) REPLACING......
MOVE RATE-TABLE TO OUTPUT-AREA.
```

If a data item is repeated (i. e. , involves the OCCURS clause at its own or a higher level), then the name of this item must be subscripted whenever it is referenced. Furthermore, a data-name can only be subscripted if the data item is repeated.

SECTION II
REFERENCE FORMAT

The COBOL B source program is written on the Honeywell COBOL Programming Form (#1523 or #2235). It is from these forms that the source program card deck is punched. Since the placing of information in specific card columns is imperative, card column numbers are shown on these forms.



Honeywell
ELECTRONIC DATA PROCESSING

COBOL PROGRAMMING FORM

PUNCHING INST.

CODED

PUNCH

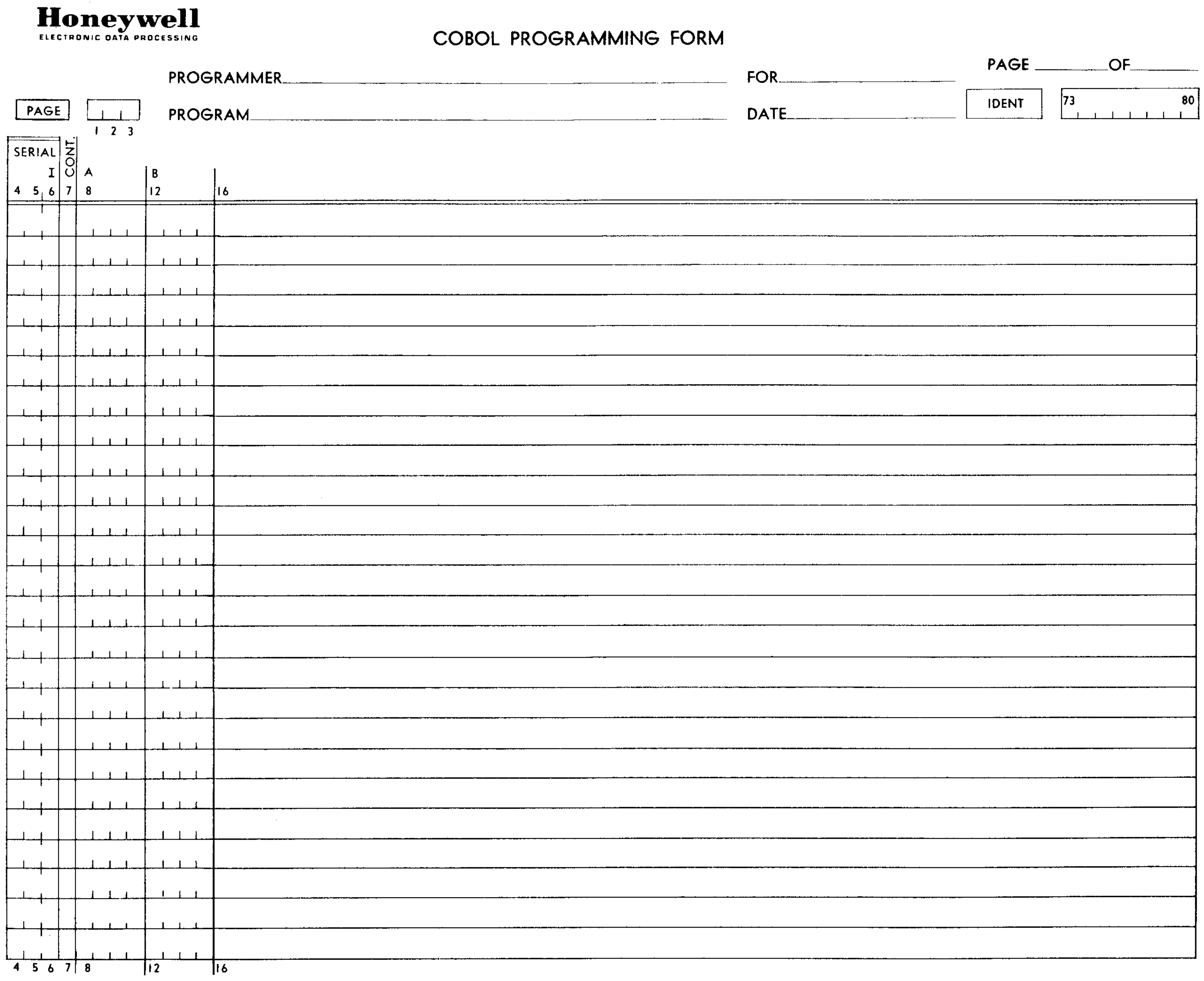
CARD FORM NO.

PAGE ____ OF ____

PROGRAM ____ PROGRAMMER ____ DATE ____ REV. NO. ____

SEQUENCE		C O N T	A	B		IDENTITY
PAGE	LINE					
1 2 3	4 5 6	7	8	12	16 20 24 28 32 36 40 44 48 52 56 60 64 68	73 80

At the head of the form there are numerous blanks which serve program documentation purpose only (i.e., PROGRAM _______, PROGRAMMER _______, DATE _______, REV. NO. _______). None of this information is punched into the source-language card deck.



Honeywell
ELECTRONIC DATA PROCESSING

COBOL PROGRAMMING FORM

PROGRAMMER ______ FOR ______ PAGE ____ OF ____

PAGE | 1 2 3 | PROGRAM ______ DATE ______ IDENT | 73 80

SERIAL 4 5 6	CONT. 7	A 8	B 12	16

4 5 6 7 8 12 16

1523 REV. 1 8-62

CARD FORMAT

The card format consists of four fields:

1. Sequence number field
2. Area A
3. Area B
4. Remarks field

NOTE: Column 7 is never used with COBOL Compiler B.

Sequence Number Field

The sequence number field (columns 1-6) may contain any of the 64 legal punches (see "Character Set," Section I). If the sequence number field of the first card contains all blanks, the compiler ignores the sequence number field on all cards in the deck. If the sequence number field of the first card contains any legal punch or punches, a sequence check is performed by the compiler and a warning diagnostic is issued for each sequence number not greater than the preceding sequence number.

Area A

All division, section, and paragraph headers, and FD and 01 level indicators must start in area A (columns 8-11).

Area B

All other text, not falling within the definition of area A, must start in area B (columns 12-72).

Remarks Field

The remarks field (columns 73-80) is ignored by the compiler but reproduced on the output listing.

Continuation of Source-Coding Line

When it is necessary or desired to continue a line of source coding from one coding-form line to another, the following rule applies:

> To continue a source-coding line on the succeeding form line, the first continuing word on the second line must begin under Area B (column 12). As many spaces as desired can follow the last word on the first line, or the last word of the first line can end at column 72. In any case, a word cannot be broken or hyphenated. The compiler issues a warning diagnostic if column 7 contains anything other than a blank.

KEYPUNCHING THE SOURCE PROGRAM

If the COBOL source program is written on a "free-form" coding sheet, it is important that the keypunch operator as well as the programmer work within a basic set of rules so that the source program deck is as free from error as possible. For the programmer, these rules consist for the most part of the avoidance of ambiguities in his writing and the observance of the reference format rules given in the first part of this section. The rules for keypunching from the coding sheet are given below, under "Keypunching Conventions." While individual installations may have their own special conventions, the following basic rules are suggested.

Programming Conventions

1. The source program should be printed if the programmer's handwriting is not legible to the keypunch operator.
2. To avoid confusing certain alphabetic characters with numeric digits, and for clarity for the keypunch operator, the following conventions should be followed:

0	Alphabetic	- written as O.
	Numeric	- written as Ø.
1	Alphabetic	- written as I.
	Numeric	- written as 1.
S	Alphabetic	- written with tails to distinguish from numeric 5.
T	Alphabetic	- written with a clearly defined crossbar to distinguish from numeric 7.
D	Alphabetic	- written with a straight leading line to distinguish from alphabetic O.
Z	Alphabetic	- written with a slash (Ƶ) to distinguish from numeric 2.
G	Alphabetic	- written with a clearly defined crossbar to distinguish from numeric 6.
U	Alphabetic	- written with a clearly rounded base to distinguish from alphabetic V.
V	Alphabetic	- written with a clearly pointed base to distinguish from alphabetic U.
E	Alphabetic	- written with a clearly separated bar to distinguish from alphabetic B.
B	Alphabetic	- written with clearly joined rounds to distinguish from alphabetic E.
SPACE SYMBOL		- written as Δ. While this symbol does not always have to occur in a source program to indicate the presence of a space or a blank, it should be used whenever the exact number of spaces to be keypunched is critical, such as in non-numeric literals.

3. The reference format for the beginning of entries or statements should be strictly adhered to.

Keypunching Conventions

1. The entry in the sequence number field is punched in columns 1-6 and may be any of the 64 legal punches.

2. In no case may there be an entry in column 7.

3. The remaining entries on a line are punched beginning in the column indicated on the programming form (column 8, 12, or 16) and continue through column 72. In no case can the line be continued into column 73 and beyond.

4. To a COBOL programmer, a "word" can be composed of other than the alphabetic characters A through Z. It is in the sense of a string of one or more of any characters except a space that "word" is used in the remainder of this discussion. Line entries (other than those already discussed) consist of one or more words. Unless a word is followed by a punctuation mark, it must be followed by a space. Any punctuation mark immediately following a word must itself be followed by a space. (It must be noted, however, that a hyphen can never occur as a punctuation mark following a word and that unless a hyphen is included between quotation marks, it can never be succeeded by a space but must be followed by another character. Also, unless a hyphen is used to indicate a negative quantity, such as -187.35, it can never be preceded by a space.) Any punctuation mark immediately following a word must be followed by a space. These spaces following a word or a punctuation mark can be omitted if the last character of the word or the punctuation mark is punched in column 72. A left parenthesis can never be followed by a space if it is not included within quotation marks.

   A beginning quotation mark must not be followed by a space, and an ending quotation mark must not be preceded by a space unless such a space has been indicated by the programmer. It is especially important that the number of spaces within a word bounded by quotation marks be exactly the number of spaces indicated by the programmer.

5. The line entry is punched as indicated by the programmer's groupings of words. When the last character of the entry is punched, any remaining columns on the card through column 72 are left blank. If the whole content of the line cannot be punched on one card, it must be continued on another card according to the following rules:

   > When the last letter or character of a word is punched in column 72, the first character (other than a space following that word) is punched in column 12. Columns 7 through 11 are left blank. The rest of the line being continued is punched through column 72 if necessary. Should the punching of the line still not be completed, it can be extended to another card in the same manner.

SECTION III

THE IDENTIFICATION DIVISION

The purpose of the IDENTIFICATION DIVISION is to identify the source program and the outputs of a compilation. In addition, the programmer can include certain other information relative to the program for documentation purposes.

DIVISION FORMAT

IDENTIFICATION DIVISION. If a serial number appears on this card, a card-number sequence check occurs.

PROGRAM-ID. program-name . (6 characters maximum; any characters in the complete COBOL B set can be used except the hyphen).

Other documentation statements may follow. These statements are accepted by the compiler but have no effect on the program.

SECTION IV
THE ENVIRONMENT DIVISION

The ENVIRONMENT DIVISION specifies the computer on which the source program is to be compiled, the computer on which the object program is to be executed, the mnemonic names assigned by the programmer to specific pieces of hardware referenced in the source program, the source program files assigned to hardware units, and the input/output techniques to be applied to the files when the object program is executed.

There are two sections in the ENVIRONMENT DIVISION.

1. CONFIGURATION SECTION. This section deals with the overall specifications of the source and object computer and is divided into three paragraphs:
   a. SOURCE-COMPUTER.
   b. OBJECT-COMPUTER.
   c. SPECIAL-NAMES (optional).
2. INPUT-OUTPUT SECTION. This section deals with the definition of the external media and contains information needed to create the most efficient transmission and handling of data between the media and the object program. This section is divided into two paragraphs:
   a. FILE-CONTROL.
   b. I-O-CONTROL.

STRUCTURE

The source program ENVIRONMENT DIVISION begins with the heading:

```
ENVIRONMENT DIVISION.
```

Each section within this division begins with the appropriate section name followed by the word SECTION, and each paragraph within each section begins with the appropriate paragraph-name:

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

The sections and the paragraphs within the sections must appear in the source program in the order outlined above.

FORMAT AND ENTRIES IN THE ENVIRONMENT DIVISION

CONFIGURATION SECTION

FUNCTION: To define hardware assignments.

FORMAT:

```
CONFIGURATION SECTION.

                                                  ┌  120 ┐
  SOURCE-COMPUTER.  [ SERIES-200 ]  MODEL  -     │  200 │
                                                 <  1200 >
                                                  └ 2200 ┘

     [ , MEMORY SIZE  integer-1  CHARACTERS ]  .

                                                  ┌  120 ┐
                                                  │  200 │
  OBJECT-COMPUTER.  [ SERIES-200 ]  MODEL  -     <  1200 >
                                                  └ 2200 ┘

     [ , MEMORY SIZE  { ADDRESS integer-2 THRU integer-3 } ]
                      { integer-4 CHARACTERS             }

     [ , ASSIGN OBJECT-PROGRAM TO  { CARD-READER } ]
                                   { TAPE-UNIT   }

     [ , SINGLE-BUFFER ]  [ , WITH  EDIT-OPTION ]  .
```

TECHNICAL NOTES:

1. This section must always appear in the source program.
2. Both the SOURCE-COMPUTER and the OBJECT-COMPUTER paragraphs must appear in this section in the format given above.
3. The individual paragraphs, including the optional SPECIAL-NAMES paragraph, are described in the following pages.

SOURCE-COMPUTER

FUNCTION: To describe the computer upon which the source program is to be compiled.

FORMAT:

$$\underline{\text{SOURCE-COMPUTER.}}\ \left[\underline{\text{SERIES-200}}\right]\ \underline{\text{MODEL}} - \left\{\begin{array}{c}\underline{120}\\ \underline{200}\\ \underline{1200}\\ \underline{2200}\end{array}\right\}$$

$$\left[,\ \underline{\text{MEMORY}}\ \text{SIZE}\ \ \text{integer-1}\ \ \underline{\text{CHARACTERS}}\right]\underline{.}$$

TECHNICAL NOTES:

1. This paragraph must appear in the source program.
2. The model number must be specified. It is used for documentation purposes only.
3. If the MEMORY SIZE is specified, integer-1 may be any number (of CHARACTERS). This clause is used for documentation purposes only and has no effect on the actual memory used by the source computer. If it is desired that more memory be used, see Section VIII, "Operating Instructions."

OBJECT-COMPUTER

FUNCTION: To describe the computer upon which the object program is to run.

FORMAT:

```
                                                 ┌ 120  ┐
OBJECT-COMPUTER.  [ SERIES-200 ]  MODEL-  <  200   >
                                                 │ 1200 │
                                                 └ 2200 ┘

    [, MEMORY SIZE { ADDRESS integer-2 THRU integer-3 } ]
                   { integer-4 CHARACTERS             }

    [, ASSIGN OBJECT-PROGRAM TO { CARD-READER } ]
                                { TAPE-UNIT   }

    [, SINGLE-BUFFER ]  [, WITH EDIT-OPTION ] .
```

TECHNICAL NOTES:

1. This paragraph must appear in every source program.

2. The model number must be specified. It is used for documentation purposes only.

3. In the MEMORY SIZE clause, there are two options:

   a. If the

      ADDRESS integer-2 THRU integer-3

      option is specified, integer-2 must be the address of the low-order character position of high-speed memory that can be used by the object program.

   b. If the

      integer-4 CHARACTERS

      option is specified, integer-4 may be any number.

4. If the MEMORY SIZE clause is not specified, it is assumed that the source program operates within 8192 characters of high-speed memory.

5. The object program is normally produced on cards. If it is desired to produce the object program on tape, the clause

   ASSIGN OBJECT-PROGRAM TO TAPE-UNIT

   must be specified.

6. The SINGLE-BUFFER option directs the compiler to apply single buffering to every file. If this option is not specified, double buffering is applied to every file.

   Exception: If SINGLE-BUFFER is specified, it is still possible to designate one file assigned to a terminal device (printer or reader/punch) as being double buffered. See the APPLY DOUBLE-BUFFER option in the I-O-CONTROL paragraph.

7. WITH EDIT-OPTION must be specified if editing is desired as no edit subroutine is provided by the compiler. This option also indicates that Editing Instructions are included in the machine configuration.

SPECIAL-NAMES

FUNCTION: To provide a means of relating hardware units with mnemonic-names and to relate the status of program switches with switch-status-names.

FORMAT:

$$\left[\underline{\text{SPECIAL-NAMES.}}\quad \left[\left\{\begin{array}{l}\underline{\text{PAGE}}\\ \underline{\text{CHANNEL a}}\end{array}\right\}\ \underline{\text{IS}}\ \text{mnemonic-name}\right]\right.$$

$$\left.\left[\underline{\text{SENSE-SWITCH m}}\ \left\{\begin{array}{l}\underline{\text{ON}}\\ \underline{\text{OFF}}\end{array}\right\}\ \text{STATUS}\ \underline{\text{IS}}\ \text{switch-status-name}\right]\ \underline{.}\ \right]$$

TECHNICAL NOTES:

1. This paragraph is not required if mnemonic-names and sense-switch-status names are not used in the source program.
2. The assigned mnemonic-name for PAGE is used to advance the printer page to the head of form. The assigned mnemonic-name for CHANNEL a is used to advance the printer page to a position governed by channel a of the vertical format, paper-tape loop. Acceptable values for a depend on the specific printer to be used, as listed in the appropriate hardware manuals.
3. In a SENSE-SWITCH entry, m indicates the number of a switch, 1 through 4.
4. More than one CHANNEL or SENSE-SWITCH clause may be specified.

INPUT-OUTPUT SECTION

FUNCTION: To specify the external media and the information needed to effect the most efficient transmission and handling of data between the media and the object program.

FORMAT:

```
INPUT-OUTPUT SECTION.
FILE-CONTROL.
                                        ( INPUT-TAPE m  )
                                        | OUTPUT-TAPE m |
         SELECT file-name-1 ASSIGN TO  <  CARD-READER    >  .
                                        | CARD-PUNCH    |
                                        ( PRINTER       )
  [ I-O-CONTROL.
         APPLY DOUBLE-BUFFER ON file-name-2 . ]
```

TECHNICAL NOTES:

1. The SELECT clause identifies the beginning of information concerning file-name-1.

2. A SELECT clause must be specified for each file used in a source program. No more than six files may be selected for any one program. The file-name of each selected file must be unique within a program. Each file must have a file description in the DATA DIVISION of the source program.

3. If the

   APPLY DOUBLE-BUFFER

   clause is specified, the file (file-name-2) is double buffered. If SINGLE-BUFFER has been specified, DOUBLE-BUFFER serves to designate a non-tape file as being double buffered. Only one DOUBLE-BUFFER clause may be designated.

   If neither SINGLE-BUFFER nor DOUBLE-BUFFER is specified, double buffering is applied to every file.

4. If the

   APPLY DOUBLE-BUFFER

   clause is not used, omit the entire I-O-CONTROL paragraph.

SECTION V
THE DATA DIVISION

Data to be processed falls into two categories:

1. Data contained in files. Such data enters or leaves the internal memory of the computer from a specified area or areas.
2. Data which is developed internally and is placed into intermediate or working-storage.

Consequently, the DATA DIVISION consists of two sections:

1. FILE SECTION, in which files and records in files are described.
2. WORKING-STORAGE SECTION, in which memory space is allocated for the storage of intermediate results of processing.

The approach taken in defining file information is to distinguish between the physical characteristics of the file (that is, the file description) and the conceptual characteristics of the data contained within the file (that is, the record description). The physical aspects of a file include the way the logical records are grouped within the physical limitations of the file medium, the means by which the file can be identified, and the mode in which the file is recorded on its storage medium. The conceptual characteristics of a file encompass the explicit definition of each logical entity within the file itself.

For processing purposes, the contents of a file are divided into logical records (a logical record being any consecutive set of information). Several logical records can occupy a block on the storage medium.

The concept of a logical record is not restricted to file data but is carried over into the definition of working-storage and constants. Thus, working-storages and constants can be grouped into logical entities and defined by a series of record description entries.

In effect, then, two elements enter into the description of data in the DATA DIVISION:

1. The description of files.
2. The description of records.

The FILE SECTION includes both elements; the WORKING-STORAGE SECTION consists solely of the description of records.

STRUCTURE

The source program DATA DIVISION begins with the heading:

```
        DATA DIVISION.
```

Each section within the DATA DIVISION begins with the appropriate section name followed by the word SECTION:

FILE SECTION.
WORKING-STORAGE SECTION.

When a section is not required, its name need not appear.

The sections themselves consist of a series of related and unrelated entries. An entry consists of a level indicator, a data-name, and a series of independent clauses which can be separated from each other by the use of commas. The entry is terminated by a period. A file description consists of a single entry. A record description, however, consists of one or more entries.

FILE DESCRIPTION ENTRY

A file description entry contains information pertaining to the physical aspects of a file. In general, it may include the size of the physical records, the names and values of the label records contained in the file, and the names of the data records which compose the file. The listing of data and label record names in a file description entry serves as a cross-reference between the file and the records in the file.

A file description entry consists of a level indicator, a file-name, and a series of independent clauses which define the physical and logical characteristics of the file. The mnemonic level indicator FD is used to identify the start of a file description entry and distinguishes this entry from those associated with a record description.

In the following pages, the individual clause formats are arranged in alphabetic order. They are preceded by the complete entry format containing the clauses in this recommended order of appearance.

FILE DESCRIPTION

FUNCTION: To provide information concerning the physical structure, identification, and record descriptions pertaining to a given file.

FORMAT:

$$(\underline{\text{FILE SECTION.}})$$

$$\underline{\text{FD}}\ \text{file-name}$$

$$\left[,\ \underline{\text{BLOCK}}\ \text{CONTAINS integer-1}\ \left\{\begin{array}{l}\underline{\text{RECORD}}\\ \underline{\text{RECORDS}}\end{array}\right\}\right]$$

$$,\ \underline{\text{LABEL RECORDS}}\ \text{ARE}\ \left\{\begin{array}{l}\underline{\text{STANDARD}}\\ \underline{\text{OMITTED}}\end{array}\right\}$$

$$\left[,\ \underline{\text{VALUE OF IDENTIFICATION}}\ \text{IS literal-1}\right]$$

$$,\ \underline{\text{DATA}}\left\{\begin{array}{l}\underline{\text{RECORD}}\ \text{IS}\\ \underline{\text{RECORDS}}\ \text{ARE}\end{array}\right\}\ \text{record-name-1}\ \left[,\ \text{record-name-2}\ \ldots\right]\ \underline{.}$$

TECHNICAL NOTES:

1. The individual entries for the FILE SECTION are discussed in the following pages. FILE SECTION must always appear.
2. The "FD file-name" entry must be given for <u>each</u> file in the source program.
3. The

   LABEL RECORDS ARE ...

   clause must be given for each FD file-name entry.

BLOCK CONTAINS

FUNCTION: To specify the size of a physical record (i. e., a block) on magnetic tape.

FORMAT:

$$\left[,\ \underline{\text{BLOCK}}\ \text{CONTAINS integer-1} \begin{Bmatrix}\underline{\text{RECORD}}\\ \underline{\text{RECORDS}}\end{Bmatrix}\right]$$

TECHNICAL NOTES:

1. Integer-1 must be a numeric literal with an integral value.
2. This clause is required when the block is to contain more than one logical record.

DATA RECORD(S)

FUNCTION: To cross-reference the description of data records with their associated file.

FORMAT:

$$, \underline{\text{DATA}} \left\{ \begin{array}{l} \underline{\text{RECORD}} \text{ IS} \\ \underline{\text{RECORDS}} \text{ ARE} \end{array} \right\} \text{record-name-1} \left[, \text{ record-name-2} \ldots \right]$$

TECHNICAL NOTES:

1. The presence of more than one record-name indicates that the file contains more than one type of data record.
2. Conceptually, all data records within a file share the same area. This is in no way altered by the presence of more than one type of data record within the file.
3. Both record-name-1 and record-name-2 must have 01 level numbers.
4. The length of a data record cannot exceed 4,095 characters.
5. This clause is used for documentation purposes.

FD

FUNCTION: To indicate the highest level of a file description.

FORMAT:

<u>FD</u> file-name

TECHNICAL NOTES:

1. FD must be the first entry of the file description. It is entered beginning in column 8 of the coding form.
2. The file-name must be selected in the FILE-CONTROL entry of the ENVIRONMENT DIVISION.

LABEL RECORDS

FUNCTION: To cross-reference the description of label records with their associated file.

FORMAT:

$$, \underline{\text{LABEL RECORDS}}\ \text{ARE} \left\{ \begin{array}{l} \underline{\text{STANDARD}} \\ \underline{\text{OMITTED}} \end{array} \right\}$$

TECHNICAL NOTES:

1. This clause is required in every file description entry.
2. When OMITTED is specified, the file is understood to have no header label records. Exception: A tape containing (or one which is to contain) the file must have a beginning-of-tape label.
3. When STANDARD is specified, the file is understood to conform to the Series 200 label conventions.
4. Whenever STANDARD is specified, a

   VALUE OF IDENTIFICATION IS . . .

   clause must be given.
5. The STANDARD option may be used for files assigned to tape units; the OMITTED option must be used for files not assigned to tape units.

VALUE OF

FUNCTION: To specify the value of an item in the label records associated with a magnetic tape.

FORMAT:

[, <u>VALUE OF IDENTIFICATION</u> IS literal-1]

TECHNICAL NOTES:

1. This clause must be specified when the

   LABEL RECORDS ARE STANDARD

   clause is given.

2. This clause causes (1) the value stated in the non-numeric literal-1 to be placed in the label record identification field of an output file at object time, or (2) a check for the value stated in the non-numeric literal-1 in the label record identification field of an input file at object time.

3. The value stated in literal-1 must be in the range from 1 to 10 alphanumeric characters in length (space characters are allowed). If less than 10 characters are used, the field is right-filled with spaces.

RECORD DESCRIPTION ENTRY

A detailed data description consists of a set of entries, each of which defines the characteristics of a particular unit of data. With minor exceptions, each entry is capable of completely defining a unit of data. Because the COBOL detailed data descriptions involve a hierarchical structure, the contents of an entry can vary considerably, depending upon whether or not an entry is followed by subordinate entries.

In defining the lowest level of subdivision of data, the following information is required.

1. A level number which shows the relationship between this and other units of data.
2. A data-name.
3. A PICTURE clause.

A record description consists of a set of entries. Each record description entry, itself, consists of a level number, a data-name, and a series of independent clauses.

In the following pages, the individual clause formats are arranged in alphabetic order. They are preceded by the complete entry format containing the clauses in their recommended order of appearance.

RECORD DESCRIPTION

FUNCTION: To specify the particular characteristics of an item of data.

FORMAT:

$$\text{level-number} \begin{Bmatrix} \text{data-name-1} \\ \underline{\text{FILLER}} \end{Bmatrix} \left[\underline{\text{REDEFINES}} \text{ data-name-2} \right]$$

$$\left[, \underline{\text{OCCURS}} \text{ integer-1 TIMES} \right]$$

$$\left[, \underline{\text{PICTURE}} \text{ IS} \left\{ \text{any allowable combination of PICTURE characters} \right\} \right] \underline{.}$$

TECHNICAL NOTES:

1. For an explanation of the reference format used in the DATA DIVISION, see Section II.
2. The PICTURE clause must appear in the entry of an elementary item, and it must not appear in the entry of a group (or non-elementary) item.
3. The REDEFINES clause, when specified, must immediately follow data-name-1.
4. Detailed descriptions of each clause are given in the following pages.

data-name/FILLER

FUNCTION: To name the data being described, or to specify an unused portion of the logical record.

FORMAT:

$$\text{level-number} \left\{ \begin{array}{l} \text{data-name} \\ \underline{\text{FILLER}} \end{array} \right\}$$

TECHNICAL NOTES:

1. A data-name or the key word FILLER must be the first word following the level-number in each record description entry.
2. The key word FILLER is used to name a record item not otherwise referenced. Under no circumstances can a FILLER item be referenced directly.

level-number

FUNCTION: To show the hierarchy of data within a logical record.

FORMAT:

level-number

TECHNICAL NOTES:

1. A level-number is required as the first element in each record description entry.
2. A level-number may have a value of 01 through 05.
3. The level-number 01 indicates the first entry in each record description. This corresponds to the logical record on which the READ and WRITE verbs operate. The following examples illustrate the use of level-numbers.

```
01 payroll-master
```

The 01 level in a record description corresponds to the name of the record. When this name is referenced from the PROCEDURE DIVISION, the entire record, including all of its groups and fields, is made accessible.

```
02 employee-number
```

Data units with a level number higher than 01 can be either fields or groups of fields. The above example shows that employee-number is a field, since it is not further subdivided into smaller units.

```
02 employee-name
   03 last-name
   03 first-name
   03 initial
```

This example illustrates three fields at the 03 level which constitute the 02-level group employee-name. The three fields last-name, first-name, and initial can each be referenced separately, or their collective contents can be referenced by the group-name employee-name.

```
02 wage-data
   03 average-wage
      04 high-wage
      04 low-wage
      04 median
   03 present-pay
      04 class-code
         05 position-letter
         05 job-description
      04 salary
   03 review-rating
02 work-history
```

The above is a hierarchy of data description showing data named at various levels. The level-02 group wage-data includes all groups and fields up to but not including work-history. Specifically, it includes the groups average-wage and present-pay, and the field review-rating. The level-03 group average-wage includes only the three elementary fields high-wage, low-wage, and median. The level-03 group present-pay includes the group class-code and the field named salary. The level-04 group class-code includes the fields position-letter and job-description.

OCCURS

FUNCTION: To eliminate the need for separate entries to describe repeated data.

FORMAT:

[, OCCURS integer-1 TIMES]

TECHNICAL NOTES:

1. integer-1 must be a numeric literal with a positive integral value.
2. This clause cannot be specified in a record description entry that has an 01 level number.
3. The OCCURS clause is used in defining tables and other homogeneous sets of repeated data. When the OCCURS clause is used, the data-name which is the subject of this entry must be subscripted whenever used as an operand, regardless of the value of integer-1. Further, if this data-name is the name of a group item, then all data-names belonging to the group must be subscripted whenever used as operands. The data description clauses associated with an item whose description includes an OCCURS clause apply to each repetition of the item being described.
4. The repeated item cannot occur more times than would require 4,095 characters of memory.
5. The limit of 4,095 characters affixed to record size and the OCCURS clause can be circumvented by defining a large record or table as two records, as in the following example:

```
01 A
   02 B PICTURE IS XX, OCCURS 2000 TIMES
   .
   .
   .
01 C
   02 D PICTURE IS XX, OCCURS 1500 TIMES
   .
   .
   .
   MOVE B(3000)
```

6. At object time, the character positions in memory for record storage and/or for reading from or writing to an external medium are allocated as though the maximum number of item occurrences were in the record. There can be fewer occurrences of valid input data at object time, provided that the programmer (1) provides padding in the source program for the space beyond that taken up by the valid data and (2) tests the number of item occurrences.
7. There should not be more than 30 OCCURS clauses in a program.
8. A group may have any number of OCCURS clauses (to the prescribed limit) as long as no one OCCURS clause is subordinate to another OCCURS clause.

```
Examples: 02 d-n-1              }        02 d-n-1                }
             03 d-n-2 OCCURS    }           03 d-n-2 OCCURS      }
             03 d-n-3 OCCURS    }           03 d-n-3             }
                04 d-n-4        } acceptable   04 d-n-4 OCCURS   } unacceptable
                04 d-n-5        }              04 d-n-5          }
             03 d-n-6 OCCURS    }           03 d-n-6             }
```

PICTURE

FUNCTION: To show a detailed picture of the standard data format of an elementary item, the general characteristics of the item, and any special report editing.

FORMAT:

(data-name...) [, <u>PICTURE</u> IS character-string]

TECHNICAL NOTES:

1. A PICTURE clause can only be used to describe an elementary item (or field).

2. A PICTURE may consist of no more than 30 characters from the following list:

<u>Data Characters</u>

A	indicates an alphabetic character
X	indicates an alphanumeric character
9	indicates a numeric character

<u>Operational Symbols</u>

S	indicates the presence of a sign
V	assumed decimal point location

<u>Floating Characters</u>

$$	dollar sign (replaces last leading zero)
**	check protection (replaces leading zeroes)
Z	standard zero suppression (replaces leading zeroes with blanks)

<u>Insertion Characters</u>

$	dollar sign
,	comma
.	decimal point
B	blank

<u>Report Signs</u>

CR	credit
-	minus or negative sign

3. When consecutive repetitions of the same character occur, they may be replaced by that character followed by an unsigned numeric literal enclosed in parentheses. The value of the literal indicates the number of repetitions of the character. No more than 4 digits may appear within the parentheses.

Example: 9999999AAAAAA may be written as 9(7)A(6)

NOTE: A zero-value may not appear within the parentheses.

Example: 9V99(0) or Z (000) 99

It is important that the terms "implicit PICTURE character-string" and "explicit PICTURE character-string" be defined at this time.

"Implicit PICTURE character-string" indicates the abbreviated PICTURE as coded in the example above; viz., 9(7)A(6). The character-count of this implicit PICTURE character-string is eight characters.

"Explicit PICTURE character-string" indicates the actual PICTURE as it is expanded by the compiler. In the example above, 9999999AAAAAA is the explicit PICTURE character-string. The character-count of this explicit PICTURE character-string is 13 characters.

In determining the size of the item defined by the explicit PICTURE, all characters other than the operational symbols (S and V) are counted. The explicit PICTURE S9999V99 defines an item whose size is six characters.

4. The allowable characters (or symbols) which may appear in a character-string are defined and described below according to the class definition of the data item being described by the PICTURE clause and according to the associated move category of the item (see the MOVE verb in Section VI).

   a. Numeric Class Characters — a PICTURE is considered to be numeric if it contains at least one 9; S and V are the only other allowable characters and they are optional.

      9 Indicates that the character position contains a numeric character.

      S Indicates the presence of an operational sign which is not counted in the size of the data item. The PICTURE of a numeric data item can possess only one operational sign and, if specified, S must be the leftmost character of the PICTURE.

      V Indicates an assumed decimal point which does not occupy a character position and is not counted in the size of a data item. The PICTURE of a data item cannot contain more than one assumed decimal point. A low-order V in a PICTURE is considered redundant, since when V is not specified for a numeric data item, the decimal point is assumed to be to the right of the PICTURE description.

      Not more than sixty-three 9's can appear to the right of a V.

   b. Numeric-Edited Class Characters — a PICTURE is considered to be numeric edited if it contains any allowable characters with the exception of A and X.

      9 S V have been previously defined.

      B , } Indicates the insertion characters space (blank) and comma (,). Each insertion character is counted in the size of the data item but does not represent a numeric character position. The presence of zero suppression and check protection (**) indicates that suppression of leading insertion characters also takes place with associated space or asterisk replacement. A PICTURE description must not have as its rightmost PICTURE character the symbol "," unless immediately followed by one of the punctuation characters (comma or period).

      $$ ** Z } The floating characters, floating dollar sign ($$), floating check-protect (**), and floating zero-suppression (Z), "float through" the insertion characters, i. e. , if the most significant digit is immediately to the right of an insertion character, the floating replacement character replaces the insertion character.

      $ Indicates the dollar sign character. As a fixed insertion character, the "$" may appear only once in a

PICTURE, either as the leftmost character or following a minus sign as the second symbol in a PICTURE character-string.

. } Indicates the actual decimal point and is a special insertion character. The data item being edited is aligned by decimal point and the actual decimal point appears in the indicated character position. The actual decimal point, unlike the assumed point (V), is counted in the size of the data item. The symbols V and . are mutually exclusive within a PICTURE description. A PICTURE description must not have as its rightmost PICTURE character the symbol "." unless immediately followed by one of the punctuation characters (comma or period).

CR } - } Indicate the editing sign control characters credit (CR) and negative or minus sign (-). The symbol "CR" may only appear in the low-order position of a PICTURE and represents two character positions when counted in the size of the data item. If the data item chances to be positive, two spaces are displayed. The negative or minus symbol "-" may only appear in either the high- or low-order position of the PICTURE and represents one character position when counted in the size of the data item. If the data item chances to be positive, a space is displayed.

c. Non-Numeric Class Characters — a PICTURE is considered to be non-numeric if it contains alphabetic or alphanumeric characters.

(1) Alphabetic Characters

A Indicates that the character position contains an alphabetic character (letter or space).

B Indicates the insertion character space (blank) and is counted in the size of the data item. If data is moved to a data item whose PICTURE description contains the insertion character B, the alphabetic data is right justified within the receiving character positions independently of the insertion characters. This means that if the sending data item is shorter than the receiving item, the excess positions at the left end of the receiving field are space-filled; if the sending data item is longer than the receiving item, the excess characters are truncated from the left end of the sending data. The insertion characters are placed in the receiving data item regardless of the nature of the sending data item. If data whose PICTURE description contains the insertion character B is moved to another data item, each data item character position in the sending data item is considered as part of the data item during the move operation.

(2) Alphanumeric Character

X Indicates that the character position contains any character in the computer's character set. If a data item PICTURE consists entirely of any combination of X, A, and 9 other than all 9's, the PICTURE is treated as if it consisted of all X's.

d. Non-Numeric-Edited Class Characters — A PICTURE description containing at least one insertion character B and at least one of the characters X or A is considered to be an alphanumeric edited data item. The rightmost character must not be an insertion B.

5. A PICTURE must contain at least one of the following characters:

A X 9 Z

or at least a pair of one of the following characters:

$ * (indicating floating characters)

6. Only one type of floating replacement character, i.e., "$", "*", or "Z", can be used within a given PICTURE description. However, it is permissible for a single insertion $ to appear with floating Z or *. A PICTURE character "9" can never appear to the left of a floating or a replacement character.

7. The following limitations and restrictions must be carefully followed when writing a legal PICTURE description:

a. The size of any item defined by a PICTURE character-string cannot be greater than 4095. Examples of illegal size: X (4096) or A (2000) X (2000) A (1000).

b. The size of a numeric or alphanumeric item defined by a PICTURE character-string containing editing characters must not exceed 30. Examples of illegal size: Z (31) or X (25) B (5) X (5).

8. The following warnings indicate how COBOL Compiler B handles certain PICTURE descriptions:

a. If the low-order Z is followed by a comma and then a receiving character, that receiving character is replaced by Z.

PICTURE string as written:

Z,ZZZ,999

Modified to:

Z,ZZZ,Z99

b. If the low-order $ or * is followed by a comma or a B and then two receiving characters, the two receiving characters are replaced by $ or *.

PICTURE string as written:	Modified to:
$,$$$,999CR	$,$$$,$$9CR
$*,***,999	$*,***,**9
$$$B999	$$$B$$9

If the low-order $ or * is followed by a comma or B and something other than two receiving characters after that, the comma or B is replaced by a $ or *.

PICTURE string as written:	Modified to:
-*,***,9	-*,****9
$$$$,B999	$$$$$B999

(NOTE: In the last example the rule is not applied to the B following the comma and the blank remains; it will not be floated "into".)

c. If the low-order $ or * is preceded by a comma or a B and followed by a receiving character, that receiving character is replaced by a $ or *.

PICTURE string as written:	Modified to:
$,$$$,$99	$,$$$,$$9
***,*99	***,**9
$B$$$B$99	$B$$$B$$9

If the low-order $ or * is preceded by a comma or a B and followed by something other than a receiving character, the comma or B is replaced by a $ or *.

PICTURE string as written:	Modified to:
$$$B$9	$$$$$9
****,*9	******9
$$$,$B99	$$$$$B$$
***,*,9	******9

(NOTE: In the last two examples rule (c) was applied first, and then rule (b).)

d. If only one * appears and it is followed by a receiving character, that receiving character is replaced by an *.

Examples	Modified to:
*99,999	**9,999
-*99,999	-**9,999
$*99,999CR	$**9,999CR
-$*9,999	-$**,**9

(NOTE: In the last example, rule (b) was applied after rule (d).)

9. Non-COBOL Editing — the following items indicate editing actions contrary to CODASYL specifications:

a. No blank-when-zero in non-integral fields

Float characters to the right of the decimal point (assumed or actual) are ignored. As a result, there is no way of getting blank-when-zero in a non-integral field.

Example	Has same effect as:
$$$.$$	$$$.99
Z,ZZZVZZZ	Z,ZZZV999

b. Right justification in non-numeric editing

When a field is being moved to a non-numeric edited field, right justification with high-order space-filling or truncation is performed, rather than the left justification required by CODASYL specifications:

Literal	Receiving Picture	Receiving Result
"12"	XXBX	Δ 1 Δ 2
"1234"	XBXX	2 Δ 34

c. Space-filling in numeric edited fields

When the number of integral digits in the sending field is less than the number of integral non-edited positions in the receiving field, space-filling rather than zero-filling is used.

Literal	Receiving Picture	Receiving Result
23	$,$$9,999	ΔΔΔ $ ΔΔΔ 23
56.7	Z,Z99,999.99	ΔΔΔΔ ΔΔΔ56.70

d. No blank-when-zero in dollar edited fields

When a zero value is moved to field filled with float dollar signs, a single dollar sign will appear as the result.

Literal	Receiving Picture	Receiving Result
0	$,$$$	ΔΔΔΔ $

10. Whenever the floating or insertion dollar sign symbol ($) appears in a PICTURE character-string, it is essential that the PICTURE contain enough receiving positions apart from the leftmost dollar sign to receive all significant integral digits from the sending field.

Legal sending value	for this receiving PICTURE
0123 456	$$$$.99
01234 56	-$,$$$.99

Illegal sending value	for this receiving PICTURE
0123 456	$$$.999
012345	$,$$$
1234	$999.99

No warning is given at compilation time. If this rule is violated at object time, unpredictable results occur.

REDEFINES

FUNCTION: To allow the same storage area to contain different data items.

FORMAT:

```
        (data-name-1...) [, REDEFINES data-name-2]
```

TECHNICAL NOTES:

1. The REDEFINES clause, when specified, must immediately follow

   level-number   data-name-1

   that is, it must precede a PICTURE clause or any other data description entry.

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

3. Redefinition starts at data-name-2 and ends when a level-number less than or equal to that of data-name-2 is encountered.

4. When the level-number of data-name-2 is other than 01, it must specify a storage area of the same size as data-name-1, except in the WORKING-STORAGE SECTION.

5. The entries which give the new description of the storage must immediately follow the entries which describe the area being redefined.

6. This clause must not be used for logical records associated with the same file. The DATA RECORDS clause in the file description is used instead.

7. Subscripting of data-name-2 is not permitted.

8. The entries giving the new description of the storage area must not contain any VALUE clauses, except in condition-name entries.

9. A REDEFINES clause cannot be subordinate to, or at the same level as, an OCCURS clause.

DATA DIVISION SECTION ENTRIES

FILE SECTION

The FILE SECTION contains a section header, file description entries, and record description entries. The order of information is as follows:

```
FILE SECTION.
     FD file-name...
        01 label-name...
            .
            .
        01 record-name...
            .
            .
     FD file-name...
            .
            .
```

WORKING-STORAGE SECTION

Working-storage is that part of computer memory set aside for intermediate processing of data. The WORKING-STORAGE SECTION must not use more than 8,192 locations. Working-storage deals with computer memory requirements for the storage of intermediate data results, whereas file storage deals with the characteristics of the entire file, as well as the computer memory requirements for the storage of each record of the file.

While the FILE SECTION is composed of file description entries and record description entries, the WORKING-STORAGE SECTION is composed only of record description entries. The WORKING-STORAGE SECTION begins with a section header and a period, followed by record description entries for working-storage items, and then by record description entries for working-storage records, in that order. The skeletal format for the WORKING-STORAGE SECTION is as follows:

```
WORKING-STORAGE SECTION.
    01 data-name-1
             .
             .
             .
    01 data-name-n
    01 data-name-2
       02 data-name-3
             .
             .
             .
    01 data-name-4
       02 data-name-5
          03 data-name-6
    01 data-name-7
    01 data-name-n
```

The following record description clauses are required in each entry:

1. level-number
2. $\left\{\begin{matrix}\text{data-name}\\\text{FILLER}\end{matrix}\right\}$
3. PICTURE

In addition, there are three optional clauses which may be used in each entry:

1. REDEFINES
2. OCCURS $\left.\begin{matrix}\\\\\end{matrix}\right\}$ one or the other but not both
3. VALUE

Data elements in working-storage which bear a definite relationship to one another must be grouped into records according to the rules for formation of record descriptions. All clauses which are used in normal input or output record descriptions can be used in a working-storage record description.

The initial value of any item in the WORKING-STORAGE SECTION may be specified by using the VALUE clause of the record description. All the rules for the expression of literals and figurative constants apply. The size of a literal used to specify an initial value can be equal to or less than the size specified in the PICTURE clause of the associated data entry, but it cannot be greater.

Since level-number, $\left\{\begin{matrix}\text{data-name}\\\text{FILLER}\end{matrix}\right\}$, PICTURE, REDEFINES, and OCCURS have been discussed above under "Record Description Entry," only the VALUE clause is described in detail here.

VALUE

FUNCTION: To define the initial value of a working-storage item.

FORMAT:

$(\text{data-name-1}\ldots)\ \left[,\ \underline{\text{VALUE}}\ \text{IS literal}\right]$

TECHNICAL NOTES:

1. A VALUE clause has no meaning when applied to an input or output file record.
2. The VALUE clause may not be stated in an entry which contains an OCCURS clause or which is subordinate to an entry containing an OCCURS clause.
3. If the VALUE clause is used in an entry at the group level, the group area is initialized without consideration for the individual elementary or group items contained within this group. Further VALUE clauses cannot be stated at the subordinate levels within the group.
4. When VALUE is not specified, the initial contents of the working-storage area is zero.

SECTION VI
THE PROCEDURE DIVISION

The PROCEDURE DIVISION contains all procedures needed to solve a given problem. These are written as sentences and combined to form paragraphs under paragraph-names.

Every word in a source program PROCEDURE DIVISION must be one of the following:

1. A Series 200 COBOL reserved word.
2. A word previously described (i.e., defined) in the DATA or the ENVIRONMENT DIVISION.
3. A paragraph-name not used in any other division.
4. A figurative constant (ZERO, SPACE, and QUOTE),

The only other possible PROCEDURE DIVISION entries in the source program are:

1. Numeric literals.
2. Non-numeric literals.
3. Punctuation marks.

The first entry in the PROCEDURE DIVISION of the source program must be the words PROCEDURE DIVISION. The next entry must be a paragraph-name.

COBOL procedures are expressed in a manner similar (but not identical) to normal English prose. The basic unit of procedure formation is a sentence or a group of successive sentences. A procedure is a paragraph or a group of successive paragraphs within the PROCEDURE DIVISION. A source program must contain at least one paragraph-name in the PROCEDURE DIVISION.

A PROCEDURE DIVISION sentence is made up of verb statements. Each statement must follow a given format (described in the following pages). While the format may differ for different verbs, the verb must be the first word of each statement. For COBOL purposes, the word IF is regarded as a verb since it results in the generation of coding. The verb statements can be categorized into three types, as follows:

1. Imperative
   a. Arithmetic

      ADD
      SUBTRACT
      MULTIPLY
      DIVIDE

b. Sequence Control (Branching)

   GO TO
   ALTER
   PERFORM

c. Data Movement

   MOVE

d. Ending

   STOP

e. Input/Output

   ACCEPT
   CLOSE
   DISPLAY
   OPEN
   READ
   WRITE

2. Conditional

   IF

3. Compiler-Directing

   EXIT
   NOTE

The verbs are discussed individually in alphabetic order under "PROCEDURE DIVISION Verb formats and Verb Descriptions" below.

STATEMENTS

An imperative statement consists of either a verb (excluding IF, EXIT, and NOTE) and its operands or a sequence of imperative statements. A GO TO or a STOP statement which appears in a sequence of imperative statements must be the last statement in the sequence. A conditional statement has the following form:

IF condition imperative-statement

A compiler-directing statement consists of a compiler-directing verb and its operands.

SENTENCES

A sentence consists of a sequence of one or more statements, the last of which is terminated by a period. The statements composing the sentence must be either one compiler-directing statement or one or more imperative or conditional statements, syntactically correct according to the above rules. A sentence which is composed of an imperative or a conditional statement(s) is called a procedural sentence.

Imperative Sentences

An imperative statement terminated by a period is an imperative sentence.

EXAMPLE: MOVE A TO B.
MOVE A TO B, ADD C TO D.

An imperative sentence can contain either an unconditional GO statement or a STOP RUN statement, which (if present) must be the last statement in the sentence.

EXAMPLE: MOVE A TO B, ADD C TO D, GO TO START.

Conditional Sentences

A conditional statement terminated by a period is a conditional sentence.

EXAMPLE: IF X IS EQUAL TO Y MOVE A TO B, MOVE
C TO D.

Compiler-Directing Sentences

A compiler-directing statement terminated by a period is a compiler-directing sentence. For example,

EXIT.

Sentence Execution

For the remainder of this discussion, the phrase "execution of a sentence" (or a statement within a sentence) is interpreted to mean "execution of object program coding compiled from a sentence (or from a statement within a sentence) which has been written in COBOL." The phrase "transfer of control" is interpreted to mean "transfer of control in the object program by transferring (going) from one place (control point) to another place (control point) out of the written sequence." The phrase "passing of control" is interpreted to mean "passing of control in the object program by passing from one place (control point) to the next place (control point) in the written sequence."

Whenever a GO TO statement is encountered during the execution of a sentence or a statement, there is an unconditional transfer of control to the first procedural sentence of the paragraph referenced by the GO TO statement.

An imperative sentence is executed in its entirety and control is passed to the next procedural sentence (unless it consists of an unconditional GO TO).

The following four elements form a skeletal conditional sentence.

<u>IF</u> condition imperative-statement <u>.</u>

The condition is an expression which is either true or false. If the condition is true, then the statement is executed and control is transferred to the sentence. If the condition is false, control is passed to the next sentence.

Example: IF A IS EQUAL TO B GO TO C. MOVE X TO Y.

In the example, if A is equal to B, control is transferred to paragraph C. If A is not equal to B, control is transferred to the next sentence (MOVE...).

Compiler-directing sentences direct the compiler to take action at compilation time. Procedural sentences, on the other hand, denote action to be taken by the object program.

Control Relationship Between Procedures

In COBOL, imperative and conditional sentences describe the procedure that is to be accomplished. The sentences are written successively, according to the reference format, to establish the sequence in which the object program is to execute the procedure.

In the PROCEDURE DIVISION, names are used so that one paragraph can reference another by name. In this way, the sequence in which the object program is to be executed can be varied simply by transferring to a named paragraph.

In executing procedures, control is transferred only to the beginning of a paragraph. Control is passed to a sentence within a paragraph only from the sentence written immediately preceding it. If a paragraph is named, control can be passed to it from the sentence immediately preceding it, or can be transferred to it from any sentence which contains a GO TO or PERFORM statement followed by the name of the paragraph to which control is to be transferred.

PARAGRAPHS

So that the source programmer can group several sentences to convey one idea (procedure), paragraphs have been included in COBOL. In writing procedures according to the rules of the PROCEDURE DIVISION and the requirements of the reference format, the source programmer begins a paragraph with a name, which consists of a word followed by a period.

The source programmer usually puts compiler-directing sentences in their own paragraphs. Paragraphs composed of compiler-directing sentences are called "compiler-directing paragraphs." Paragraphs which contain at least one procedural sentence are called procedural paragraphs.

CONDITIONALS

Conditional procedures are one of the keystones in describing data processing problems. COBOL makes available to the programmer several means of expressing conditional situations. COBOL conditionals generally involve the key word IF, followed by the conditions to be examined, followed by the operations to be performed. Depending upon the truth or falsity of the conditions, different sets of operations are to be performed.

PROCEDURE DIVISION VERB FORMATS AND VERB DESCRIPTIONS

In the following pages the PROCEDURE DIVISION verbs and the format or formats associated with each are discussed. The verbs are arranged alphabetically for the purpose of this discussion.

ACCEPT

FUNCTION: To receive low-volume data from a peripheral device.

FORMAT:

<u>ACCEPT</u> data-name

TECHNICAL NOTE:

The ACCEPT device is the card-reader. If the size of the field defined by data-name is greater than 80 characters, only one card is read, and only the first 80 character positions of the field defined by data-name are filled. The remaining positions are left unchanged.

ADD

FUNCTION: To add two numeric data items and set the value of an item equal to the result.

FORMAT:

$$\underline{\text{ADD}} \begin{Bmatrix}\text{literal-1}\\ \text{data-name-1}\end{Bmatrix} \left[\underline{\text{TO}}\right] \begin{Bmatrix}\text{literal-2}\\ \text{data-name-2}\end{Bmatrix} \left[\underline{\text{GIVING}}\text{ data-name-3}\right]$$

$$\left[\underline{\text{ROUNDED}}\right]\left[\text{, ON }\underline{\text{SIZE ERROR}}\text{ imperative-statement}\right]$$

TECHNICAL NOTES:

1. Each ADD verb statement must contain two operands (viz., an addend and an augend).
2. Only numeric fields and numeric literals can be used as operands in ADD verb statements. The only figurative constant that can be used is ZERO.
3. A data-name can only reference an elementary item. Neither the addend nor the augend can contain editing symbols. Data-name-3 (GIVING option) may be edited.
4. Each operand can have an operational sign and an implied decimal point. There is no practical limit to the length of operands in an ADD statement. If the decimal point is not indicated for an ADD operand, it is assumed to be to the right of the least significant digit of the operand.
5. Literals cannot be used to receive the sum.
6. The sum is stored as follows (unless ON SIZE ERROR results in detection of a size error):
   a. When neither TO nor GIVING occurs in the ADD statement, the sum is stored in the receiving field data item of the statement.
   b. When TO occurs in the ADD statement but GIVING does not occur, the value of the operand preceding the TO is added to the receiving field.
   c. When GIVING occurs in the ADD statement, the values of the operands preceding the GIVING are added together, and the sum obtained is stored in the data item following the GIVING.
7. Operands are aligned according to implied decimal points. Zeros are right-filled as necessary; that is, 99∧9 and 99∧999 are aligned and zero-filled:

   99∧900
   99∧999
8. Truncation of right-hand digits of the sum can occur during the storage of that sum according to the size associated with the receiving field. In general, when the receiving field contains fewer decimal places than the sum, right truncation occurs.
9. SIZE ERROR and ROUNDED options are provided for those cases where truncation may take place.

a. When the SIZE ERROR option is specified, a test is made at object time to see if overflow occurs when the sum is stored in the receiving field. If overflow occurs, the sum is not stored. Instead, the "imperative-statement" associated with the SIZE ERROR option is performed.

b. When the ROUNDED option is specified, a test is made at compilation time to see if right truncation occurs when the sum is stored in the receiving field. If right truncation occurs, the least significant digit that can be stored is increased by one if the most significant digit truncated is in the range 5 through 9.

Example: Store 974∧617 in 999.99 Result: 974.62

c. When the SIZE ERROR option is used in conjunction with the GIVING option, the following restriction applies: If data-name-3 contains any editing other than zero suppression, SIZE ERROR handling is unspecified.

10. Note should be taken of the optional use of the words TO and GIVING. For example,

ADD A TO B is the logical equivalent of A + B = B

ADD A TO B GIVING C is the logical equivalent of A + B = C

11. The ability to use both TO and GIVING in the same ADD statement, while equivalent to common English usage, may not be allowed in other manufacturers' COBOL compilers.

ALTER

FUNCTION: To modify a predetermined sequence of operations.

FORMAT:

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

TECHNICAL NOTE:

Paragraph-name-1 must be the name of a paragraph which contains a single sentence consisting of:

```
GO TO paragraph-name.
```

CLOSE

FUNCTION: To terminate the processing of input and output files and to provide a closing convention associated with these files.

FORMAT:

<u>CLOSE</u> file-name [WITH <u>NO REWIND</u>]

TECHNICAL NOTES:

1. A CLOSE of a file assigned to a terminal device (card reader, card punch, or printer) causes no action.
2. The resulting actions for tape files are as follows:

   For input files the reel is rewound unless NO REWIND is specified.

   For output files, the actions are as follows:

   a. If the BLOCK CONTAINS integer is other than 1 and any unwritten records remain in the final block, the unused record positions in the final block are set to the padding character (77_8) and the block is written.
   b. The label area is reset to spaces.
   c. If LABEL RECORDS ARE STANDARD is specified, the appropriate fields in the label area are overlayed with "1EOFΔ" and the VALUE OF IDENTIFICATION literal, and the 80-character label is written.
   d. Two "1ERIΔ" blocks are written and the tape is back-spaced twice.
   e. If NO REWIND has not been specified, the tape is rewound.

DISPLAY

FUNCTION: To display low-volume data.

FORMAT:

$$\underline{\text{DISPLAY}} \begin{Bmatrix} \text{literal-1} \\ \text{data-name-1} \end{Bmatrix}$$

TECHNICAL NOTE:

The DISPLAY device is the printer. The field specified is printed on one line of the printer page. If the field exceeds the size of the printer line, the rightmost data is truncated.

DIVIDE

FUNCTION: To divide one numerical data item into another and set the value of an item equal to the result.

FORMAT:

$$\underline{\text{DIVIDE}} \left\{ \begin{matrix} \text{literal-1} \\ \text{data-name-1} \end{matrix} \right\} \underline{\text{INTO}} \left\{ \begin{matrix} \text{literal-2} \\ \text{data-name-2} \end{matrix} \right\} \left[\underline{\text{GIVING}} \text{ data-name-3} \right]$$

$$\left[\underline{\text{ROUNDED}} \right] \left[\text{, ON } \underline{\text{SIZE ERROR}} \text{ imperative-statement} \right]$$

TECHNICAL NOTES:

1. Each DIVIDE statement must contain two operands (viz., a dividend and a divisor).
2. Only numeric fields and numeric literals can be used as operands in a DIVIDE statement.
3. Neither the dividend nor the divisor can contain editing symbols. Only data-name-3 (GIVING option) may be edited.
4. Each operand can have an operational sign and an implied decimal point. There is no practical limit to the length of operands in a DIVIDE statement.

   If the decimal point is not indicated for a DIVIDE operand, it is assumed to be to the right of the least significant digit of that operand.
5. Division by zero constitutes a special type of size error and the rules specified under 7., below, apply.
6. Truncation of right-hand digits of a quotient can occur during the storage of that quotient according to the size associated with the receiving field. At the completion of a DIVIDE statement, the result is moved to the receiving field according to the rules of numeric MOVE statements.
7. SIZE ERROR and ROUNDED options are provided for those cases where truncation may take place.

   a. When the SIZE ERROR option is specified, a test is made at object time to see if overflow occurs when the quotient is stored in the receiving field. If overflow occurs, the quotient is not stored. Instead, the "imperative-statement" associated with the SIZE ERROR option is performed.

   b. When the ROUNDED option is specified, two more digits are developed than are needed in the result field. Then, before testing for SIZE ERROR (if specified), and before storage in the receiving field, the third rightmost digit of the result is increased by 1 if the second rightmost digit of the result is in the range 5 through 9.

   c. When the SIZE ERROR option is used in conjunction with the GIVING option, the following restriction applies: If data-name-3 contains any editing other than zero suppression, SIZE ERROR handling is unspecified.

END∆COBOL

FUNCTION: To signal the physical end of the input deck.

FORMAT:

END∆COBOL

TECHNICAL NOTE:

This entry must be placed in columns 8 through 16 and must be the last entry in every source program.

EXIT

FUNCTION: To furnish an end point for a loop, when required.

FORMAT:

```
EXIT.
```

TECHNICAL NOTES:

1. EXIT must be preceded by a paragraph-name and appear as a single, one-word paragraph. For example,

   LOOP-OUT. EXIT.

2. EXIT is used in conjunction with procedures referenced by the PERFORM verb. When a paragraph consisting only of the single verb EXIT is named as the end of range of the PERFORM, a variety of exits from the procedure can be obtained by making each point at which exit is required a transfer to the EXIT paragraph.

   When an EXIT paragraph is encountered at a time when no PERFORM statement is in effect, control passes from the paragraph preceding the EXIT paragraph to the paragraph following it.

GO TO

FUNCTION: To depart from the normal sequence of procedures.

FORMAT:

<u>GO TO</u> paragraph-name

TECHNICAL NOTE:

The paragraph-name assigned to the GO TO paragraph-name statement is referred to by the ALTER verb to modify the sequence of the program. If the GO TO paragraph-name statement is to be altered, then

1. The GO TO paragraph-name statement must immediately succeed a paragraph-name.
2. The paragraph containing the GO TO paragraph-name statement can contain only the GO TO paragraph-name statement.

IF

FUNCTION: To provide for the testing of a stated condition not connected with the ON SIZE ERROR path of an arithmetic verb (ADD, SUBTRACT, MULTIPLY, DIVIDE), or the AT END path of the READ verb statements. The determination of the truth or falsity of the condition determines the subsequent operations to be performed.

FORMAT:

$$\underline{\text{IF}} \ \text{simple-condition} \ \ \text{imperative-statement} \ \underline{.}$$

The basic format for a simple-condition clause is one of the following:

1. $$\text{data-name-1} \left\{ \begin{array}{l} \text{IS } [\underline{\text{NOT}}] \ \ \underline{\text{GREATER}} \text{ THAN} \\ \text{IS } [\underline{\text{NOT}}] \ \ \underline{\text{LESS THAN}} \\ \text{IS } [\underline{\text{NOT}}] \ \ \underline{\text{EQUAL TO}} \end{array} \right\} \left\{ \begin{array}{l} \text{data-name-2} \\ \text{literal} \end{array} \right\}$$

   This is called a relation test.

2. $$\text{data-name IS } [\underline{\text{NOT}}] \left\{ \begin{array}{l} \underline{\text{NUMERIC}} \\ \underline{\text{ALPHABETIC}} \end{array} \right\}$$

   (i. e., data-name content is not equal to all numeric class characters, or data-name content is not equal to all alphabetic class characters and/or spaces.)

3. $$[\underline{\text{NOT}}] \ \ \text{switch-status-name}$$

   A switch-status condition determines the on or off status of a hardware switch. This switch must be named in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION.

TECHNICAL NOTES:

1. A relation test specifies that at object time a test be made to determine a specified relation between two items. The only relations possible are that one item is, or is not, less than, equal to, or greater than another item.

   a. Numeric Comparisons

      A numeric comparison compares two fields, treating them as algebraic quantities. That is, the fields are aligned by decimal point, and their signs are taken into consideration in determining whether one field is greater than, equal to, or less than the other field. For a given simple relation test, a numeric comparison is performed only if both fields have numeric pictures and are elementary, and neither contains editing symbols.

   b. Non-Numeric Comparisons

      A simple relation test which does not meet the criteria specified above for numeric comparison is treated as a non-numeric comparison. In this type of test, characters from each item (beginning with the leftmost) are compared pair by pair until an inequality has been found or until the right end of one of the fields has been reached. Inequality (greater than or less than) is determined as soon as non-matching characters from corresponding character positions have been found. The determination of the relationship is based on the standard Honeywell character set (see page B-1); the lowest possible bit configuration for a character is 000000; the highest, 111111. A determination of equality is made if each of the corresponding character positions of the items contains the same bit configurations. If one of the items is shorter than the other, in effect, spaces are added to the low-order end of that item until it is the same length as the longer item.

MOVE

FUNCTION: To transfer data from one data area to another data area.

FORMAT:

$$\underline{\text{MOVE}} \begin{Bmatrix} \text{data-name-1} \\ \text{literal} \end{Bmatrix} \underline{\text{TO}} \text{ data-name-2 } \left[\underline{\text{THRU}} \text{ literal} \right]$$

TECHNICAL NOTES:

1. There are three types of MOVE operations:
   a. Figurative Constants — if literal is SPACE, QUOTE, or ZERO, either spaces or quotes or zeroes are moved into the entire receiving field.
   b. Non-Numeric — if either the sending field or the receiving field is a group item, or if the sending field is edited, or if the receiving field is not numeric, the MOVE is a non-numeric MOVE. In a non-numeric MOVE, data is filled into the receiving field from left to right. The remainder of the receiving field, if any, is space-filled. If the receiving field is too small to contain the data being moved, right truncation occurs.
   c. Numeric — if the items do not fulfill the requirements described under (b), the MOVE is considered numeric. In a numeric move, decimal-point alignment, right- or left-truncation, and/or right or left zero-fill occur when necessary.
2. If the THRU option is specified, the literal must be a one-character alphanumeric.
3. The THRU option causes a Substitute (SST) instruction to be added to the object code. The MOVE is then accomplished through the variant "literal." Only the rightmost character is moved by the Substitute instruction to the rightmost position of the receiving field.

MULTIPLY

FUNCTION: To multiply numeric data items and set the value of an item equal to the results.

FORMAT:

$$\underline{\text{MULTIPLY}} \begin{Bmatrix}\text{data-name-1}\\ \text{literal-1}\end{Bmatrix} \ \underline{\text{BY}} \begin{Bmatrix}\text{data-name-2}\\ \text{literal-2}\end{Bmatrix} \ \left[\underline{\text{GIVING}}\ \text{data-name-3}\right]$$

$$\left[\underline{\text{ROUNDED}}\right] \quad \left[\text{, ON }\underline{\text{SIZE ERROR}}\text{ imperative-statement}\right]$$

TECHNICAL NOTES:

1. Each MULTIPLY statement must contain two operands (viz., a multiplier and a multiplicand).
2. Only numeric fields can be used as operands in a MULTIPLY statement. The only figurative constant that can be used is ZERO.
3. The rightmost operand must be a data-name. Any data-name used can only reference an elementary item. Neither the multiplier nor the multiplicand can contain editing symbols. Data-name-3 (GIVING option) may be edited.
4. Each operand can have an operational sign and an implied decimal point. If the decimal point is not indicated, it is assumed to be to the right of the last significant digit of the operand. There is no practical limit to the number of digits in a MULTIPLY statement operand.
5. Literals and constant fields cannot be used to receive the product.
6. Truncation of right-hand digits of the product can occur during the storage of that product according to the size associated with the receiving field. In general, when the receiving field contains fewer decimal places than the product, right truncation occurs.
7. SIZE ERROR and ROUNDED options are provided for those cases where truncation can take place.
   a. When the SIZE ERROR option is specified, a test is made to see whether overflow occurs when the product is stored in the receiving field. If overflow occurs, the product is not stored. Instead, the "imperative statement" associated with the SIZE ERROR option is performed.
   b. When the ROUNDED option is specified, a test is made to see if right-digit truncation will occur when the product is stored in the receiving field. If right truncation occurs, the least significant digit that can be stored is increased by 1 when the most significant digit truncated is in the range 5 through 9.
   c. When the SIZE ERROR option is used in conjunction with the GIVING option, the following restriction applies: If data-name-3 contains any editing other than zero suppression, SIZE ERROR handling is unspecified.
8. Unless the

   GIVING data-name-3

   clause is used, the product is stored in data-name-2. When the GIVING option is used, the product is stored in data-name-3 which is not used in the arithmetic process. That is, MULTIPLY A BY B and MULTIPLY A BY B GIVING C are the logical equivalents, respectively, to: A · B = B and A · B = C

NOTE

FUNCTION: To provide the ability to write notes (within the PROCEDURE DIVISION) that are not compiled.

FORMAT:

<u>NOTE</u> . . . <u>.</u>

TECHNICAL NOTES:

1. A "note" may appear only within the PROCEDURE DIVISION.
2. When NOTE is the first word of a paragraph, the entire paragraph must consist of "notes" (i.e., there can be no source program coding to be compiled), since no compilation will occur until the next paragraph is reached. When an entire paragraph is a note, the paragraph must still follow the rules of proper format.
3. When NOTE is not the first word of a paragraph, only the characters between NOTE and a period followed by a space are not compiled. Compilation will begin again with the first sentence following the sentence containing NOTE.
4. Any combination of characters from the COBOL set can follow the word NOTE, except in the case of 3., above, where the occurrence of a period followed by a space is regarded as ending the "note."

OPEN

FUNCTION: To initiate the processing of input and output files, to provide the opening conventions associated with magnetic tape and punched card files.

FORMAT:

$$\underline{\text{OPEN}} \begin{Bmatrix} \underline{\text{INPUT}} \\ \underline{\text{OUTPUT}} \end{Bmatrix} \text{file-name} \left[\text{WITH } \underline{\text{NO REWIND}}\right]$$

TECHNICAL NOTES:

1. An OPEN of a card-input or card-output file causes the card device to be initialized to read or punch in special mode. An OPEN of a card-output file also causes the punching of one blank card. There is no action when a printer file is opened.

2. The actions for tape files are as follows:

Statement	Labels STANDARD	Labels OMITTED
OPEN INPUT file-name	A, B, C	A, B
OPEN OUTPUT file-name	A, B, D, F, G, H	A, B, D, F, H
OPEN INPUT file-name WITH NO REWIND	B, C	No Action
OPEN OUTPUT file-name WITH NO REWIND	E, F, G, H	No Action

<u>Table Legend:</u>

A	Rewind.
B	Read a blank into the label area.
C	Check label area for "1HDR" and correct value of ID and reel number.
D	Backspace one block.
E	Fill label area with spaces.
F	Overlay first five characters of label area with "1HDR".
G	Overlay appropriate fields in label area with value of ID and reel number.
H	Write a block from the label area.

3. An OPEN verb must be applied to a file before any READ, WRITE, or CLOSE verb is applied to it.

4. Only one OPEN can be applied to a file at any one time. That is, before a second OPEN can be executed for a file, the file must have a CLOSE executed for it.

5. The execution of an OPEN does not obtain or release the first data record of a file (a READ or a WRITE must be executed to obtain or release each data record of a file).

6. The WITH NO REWIND clause must be used if the tape is not to be automatically rewound as part of the standard opening process of the object program.

PERFORM

FUNCTION: To depart from the normal sequence of procedures in order to execute one statement, or a sequence of statements, and then return to the normal sequence.

FORMAT:

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

TECHNICAL NOTES:

1. When

   PERFORM paragraph-name-1

   is specified, only that paragraph is executed.

2. When

   PERFORM paragraph-name-1 THRU paragraph-name-2

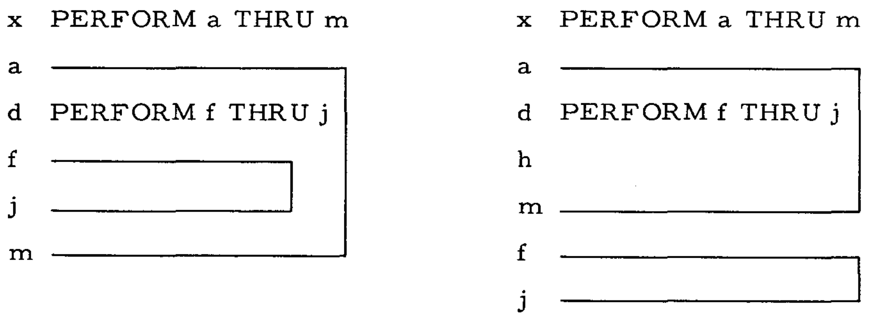
   is specified, the source-language coding from the first statement in paragraph 1 through the last statement of paragraph 2 is executed. The only necessary relation between paragraph 1 and paragraph 2 is that a sequence beginning in paragraph 1 must logically proceed into paragraph 2. When the loop specified by the PERFORM statement has more than one possible path to follow, an EXIT paragraph must provide the common ending point for each path.

3. In all cases, after the completion of a PERFORM, a bypass is automatically created around the return mechanism which has been inserted after the "last statement." Therefore, when no related PERFORM is in progress, sequence control will pass through a "last statement" to the following statement as if no PERFORM existed.

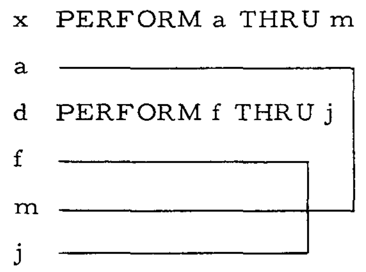
4. The programmer should note that a paragraph referenced within a PERFORM statement is also executed in the normal sequence of program execution.

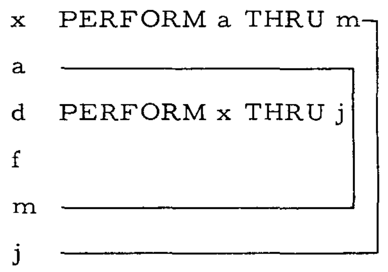
5. If a sequence of statements referenced by a PERFORM includes another PERFORM statement, the sequence associated with the included PERFORM must itself be totally included in, or totally excluded from, the logical sequence referenced by the first PERFORM.

   For example, the following illustrations are correct.



The following illustrations are incorrect:



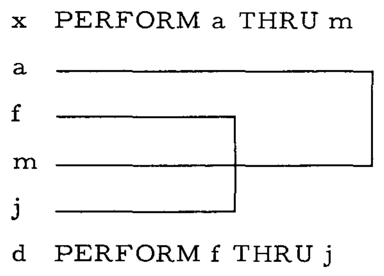


The return mechanism set by PERFORM a THRU m prevents the performance of PERFORM f THRU j in one case. In the other, a continuous loop has been set up.

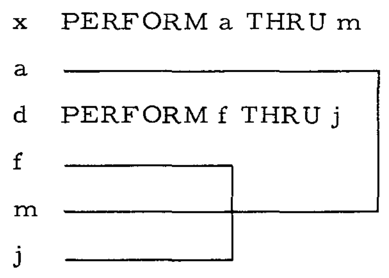
The sequence of procedures associated with a PERFORM statement can overlap or intersect the sequence associated with another PERFORM, provided that neither sequence includes the PERFORM statement associated with the other sequence.

For example:

Correct



Incorrect



READ

FUNCTION: To make the next logical record of an input file available and to allow the execution of a specified imperative statement when the end of this file is detected.

FORMAT:

<u>READ</u> file-name RECORD AT <u>END</u> imperative-statement

TECHNICAL NOTES:

1. A READ cannot be executed for a file unless the file is opened.
2. Only the information in the record made available by the current READ is accessible. (When a file consists of more than one type of logical record, these records share the same storage area.)
3. No verb in the PROCEDURE DIVISION should reference information that is not actually present in the current record. If reference is made to the nth occurrence of data which occurs fewer than n times in the accessible record, the results during object program execution are unpredictable.
4. Once the AT END path has been taken for a file, no other READ statements can be executed for that file until a CLOSE and a subsequent OPEN for that file have been executed.

STOP

FUNCTION: To halt the object program either temporarily or permanently.

FORMAT:

$$\underline{\text{STOP}} \left\{ \begin{array}{l} \underline{\text{RUN}} \\ \text{literal} \end{array} \right\}$$

TECHNICAL NOTES:

1. When the

   STOP RUN

   option is specified, the object program branches to a standard Halt instruction in the Loader. This halt can be identified by the B-address register contents being equal to 17002.

2. When the

   STOP literal

   option is specified, a program halt for operator intervention is effected. A Halt instruction is generated with the B-address register containing only the rightmost character of the literal.

   When the RUN button is depressed, the program continues with the next statement.

SUBTRACT

FUNCTION: To subtract one numeric data item from another data item and set the value of an item equal to the result.

FORMAT:

$$\underline{\text{SUBTRACT}} \left\{ \begin{matrix} \text{literal-1} \\ \text{data-name-1} \end{matrix} \right\} \underline{\text{FROM}} \left\{ \begin{matrix} \text{literal-2} \\ \text{data-name-2} \end{matrix} \right\} \left[\underline{\text{GIVING}} \text{ data-name-3} \right]$$

$$\left[\underline{\text{ROUNDED}} \right] \left[\text{, ON } \underline{\text{SIZE ERROR}} \text{ imperative-statement} \right]$$

TECHNICAL NOTES:

1. Each SUBTRACT statement must contain two operands (viz., a subtrahend and a minuend).
2. Only numeric fields and numeric literals can be used as operands in a SUBTRACT statement. The only figurative constant that can be used is ZERO.
3. Of the operands used, the rightmost must be a data-name. A data-name can only reference an elementary item. Neither the subtrahend nor the minuend can contain editing symbols. Data-name-3 (GIVING option) may be edited.
4. Each operand can have an operational sign and an implied decimal point. There is no practical limit to the length of operands in a SUBTRACT verb statement. If the decimal point is not indicated for a SUBTRACT operand, it is assumed to be logically to the right of the least significant digit of the operand.
5. Literals and constant fields cannot be used to receive the result.
6. Truncation of right-hand digits of the result can occur during the storage of that result according to the size associated with the receiving field. In general, when the receiving field contains fewer decimal places than the result, right truncation occurs.
7. SIZE ERROR and ROUNDED options are provided for those cases where truncation may take place.
   a. When the SIZE ERROR option is specified, a test is made at object time to see if overflow occurs when the result is stored in the receiving field. If overflow occurs, the sum is not stored. Instead, the "imperative statement" associated with the SIZE ERROR option is performed.
   b. When the ROUNDED option is specified, a test is made at compilation time to see if right truncation occurs when the result is stored in the receiving field. If right truncation occurs, the least significant digit that can be stored is increased by one if the most significant digit truncated is in the range 5 through 9.

      Example: Store 974‸617 in 999.99 Rounded result: 974.62
   c. When the SIZE ERROR option is used in conjunction with the GIVING option, the following restriction applies: If data-name-3 contains any editing other than zero suppression, SIZE ERROR handling is unspecified.

WRITE

FUNCTION: To release a logical record for an output file and to allow for vertical positioning if the output device is a printer.

FORMAT:

$$\underline{\text{WRITE}}\ \text{record-name}\ \left[\underline{\text{BEFORE}}\ \text{ADVANCING}\ \left\{\begin{matrix}\text{integer LINES}\\ \text{mnemonic-name}\end{matrix}\right\}\right]$$

TECHNICAL NOTES:

1. A WRITE cannot be executed for a file unless the file is open.
2. After the execution of a WRITE instruction, the associated record-name is no longer available.
3. The ADVANCING option is used for printer-line spacing control. Integer cannot specify a line advance greater than 15. The integer is interpreted modulo-16.

   Mnemonic-name must have been assigned in the SPECIAL-NAMES paragraph as PAGE IS mnemonic-name. If the output device is a printer, mnemonic-name may also be assigned to CHANNEL a in the SPECIAL-NAMES paragraph.

SECTION VII
INPUT/OUTPUT PROCEDURES

Object-time input/output operations are performed by a set of subroutines selected for inclusion in the object program according to the particular requirements of the program. These subroutines are of two types: file-specific subroutines and general subroutines.

BUFFERS

For each FD entry in the DATA DIVISION, one file-specific subroutine is produced. Associated with each file-specific subroutine is a buffer area containing one or two buffers. (Two buffers are normally allocated; however, when the clause SINGLE-BUFFER is used in the OBJECT-COMPUTER paragraph, only one buffer is allocated for each file. There is an exception to this rule: Even though the SINGLE-BUFFER clause is used, its effect can be overridden for a particular file by use of the statement APPLY DOUBLE-BUFFER TO file-name. This statement may not appear more than once, and it must refer to a file that has been assigned to a non-tape device. Thus, the object program may be arranged so that either none, one, or all of its files are double buffered. Although these three options provide increasingly greater object-time efficiency, they also require increasingly more memory space. For this reason, single buffering should be specified only for those programs which will not otherwise fit into the available memory space.

The length of the buffers or buffers allocated for a file is the length of the longest record in that file times the integer specified in the BLOCK CONTAINS clause. (Absence of a BLOCK CONTAINS clause is equivalent to the statement BLOCK CONTAINS 1 RECORD.) All records in a file are not required to be the same length; if they are of different lengths, every READ will obtain and every WRITE will release a number of characters equal to the length of the longest record in that file.

PHYSICAL FORMAT OF TAPE

Tape files are written in odd parity, fixed-sized blocks of one or more fixed-sized records. Each block is preceded by a one-character banner. Input tape files are expected to be in this format. For tape files whose BLOCK CONTAINS integer is greater than 1, the final block may contain padding records in its last position(s). A padding record is one whose first character has the value 77_8. When an output tape file is closed, sufficient padding records are automatically supplied by the general subroutine to fill the last block. For input tape files, all padding records are automatically skipped.

Regardless of whether label records are omitted or standard, the first record on a tape reel is assumed to be a tape label. If the first file on the reel has standard label records, then the first record on the reel does double duty as both a tape label and a file header label. Every tape file with standard label records is preceded by a header label and followed by a trailer label. These labels are 80-character tape blocks containing the reel number and the value of identification literal. The labels are automatically created for output files and checked for input files.

Unless the clause WITH NO REWIND is used, reels are rewound when the associated file is opened or closed. By use of the NO REWIND clause, the rewind may be suppressed; thus, the tape can be positioned to any one of several files on the same tape.

TAPE SWAPPING

If the end of tape is reached during the course of writing a file on tape, a special closing operation is initiated and the computer halts. The operator then supplies a new reel addressed the same as the previous reel and restarts the computer, and the file continues to be written on the new reel. The writer of the source program need not be concerned with this tape-swapping mechanism. A complementary process takes place when this multireel file is being read; provision is made for tape swapping when the end of the reel is reached, and reading continues with the next reel. Of course, the reels must be mounted in the same order in which they were written.

NON-TAPE FILES

For non-tape files (that is, those assigned to the card reader, the card punch, or the printer), there is no blocking, labelling, or bannering. Every logical record corresponds to a single card or print line; no header or trailer labels are expected or produced, and the first character of the record appears in the first column of the card or print line. The length of logical records in a file assigned to a terminal device is not required to be the same as the unit record length for that device (e. g. , 80 characters for card records). However, characters will be lost if the logical record length is longer than the unit record length.

I/O HALTS

When a read or write error occurs during the operation of the object program, the standard correction procedures are executed. If they are not successful in correcting the error, the computer halts with a code in the B-address register which identifies the type of halt. The codes and their meanings are listed in Table 7-1.

Table 7-1. Input/Output Device Halts at Object Time

Contents of BAR	Explanation of Halt
0001x	Read error on tape drive x
0002x	Write error on tape drive x
00110	Card read error
00120	Card punch error
00220	Printer error
0004x	Label-check failure on tape drive x
0003x	End of reel on tape drive x

The last code in Table 7-1 does not indicate an error; it directs the operator to change the reel. For each of these halts except the last, there are two options available to the operator:

1. He may push the RUN button.

   For card read, card punch, and printer errors, the program repeats the last Peripheral Data Transfer (PDT) instruction to that device. This action assumes that the operator has either

   a. retrieved the last card read from the eject pocket and reloaded it,

   b. discarded the last card punched, or

   c. manually spaced the printer and marked the last line printed as erroneous.

   For tape read or tape write errors, the program repeats the standard error procedure.

   For the tape-swap halt, it is assumed that the operator has dismounted the current reel and mounted the next reel in its place on drive x. (Alternatively, he may have already mounted the next reel on a spare drive so that the only action required is to readdress the spare drive in place of drive x.) For the incorrect label halt, it is assumed that the operator has dismounted the incorrect reel from drive x and mounted the correct reel in its place.

   All these actions except the swap are repeatable. For example, the operator may attempt rereading the same tape record any number of times simply by repetitively pushing the RUN button.

2. He may branch to the address specified by the A-address register. In all cases except the tape-swap halt, the effect of this action is that the program will ignore the error that caused the halt and proceed as if it had not occurred. Caution should be stressed in choosing this option in the case of a read or a write error halt. If a read error indication exists, the record will be accepted and processed as though no read error had occurred. If a write error indication exists, the tape is backspaced and an attempt is made to write the record correctly. If the write error persists after 63 such attempts, the record is written anyway and processing continues.

PERIPHERAL ADDRESS ASSIGNMENTS

COBOL Compiler B assumes that the standard octal addresses have been assigned to the peripheral controls as shown in Table 7-2.

Table 7-2. Standard Peripheral Address Assignments

PERIPHERAL CONTROL	OCTAL ADDRESS
Magnetic Tape	00 for output 40 for input
Card Reader	41
Card Punch	01
Printer	02

SECTION VIII

OPERATING INSTRUCTIONS

COMPILER PROGRAM OPERATING INSTRUCTIONS

1. Assign Tape Units. The self-loading tape (SLT) is always mounted on logical drive 0. A required work tape is always mounted on logical drive 1. If tape output is desired, another work tape must be mounted on logical drive 2.

2. Insert Source-Language Deck. Only one source program can be compiled at a time. The cards in the source-program deck must be in the following order:

   a. Console Call card (must have COBOLB* punched in columns 1-7)

   b. COBOL source program beginning with the

      IDENTIFICATION DIVISION card

      and ending with the

      END COBOL card

   c. One blank card

3. Press the BOOTSTRAP button once.

4. Press the RUN button twice.

5. A halt occurs with the value 17070 in the B-address register. If more than 8K characters of memory is available and the programmer desires to use it, the operator must key one of the following entries into location 0.

Keyin	Total Memory Available
2	12K
3	16K
4	20K
5	24K
6	28K
7	32K

   Regardless of whether or not a keyin is made, execution is resumed by pressing the RUN button.

   NOTE: If SENSE switch 4 is OFF, the normal printout is produced (see Section IX). If SENSE switch 4 is ON, this printout is suppressed (with the exception of the source-language diagnostics).

6. The halts that may occur during compilation are identical to those listed in Table 7-1, page 7-3.

7. At the end of compilation, a halt occurs with the B-address register containing 17002.

OBJECT PROGRAM OPERATING INSTRUCTIONS

Loading from Tape

1. Insert the Console Call card; this card must have the program-name punched in columns 1-6 and an asterisk (*) in column 7. If the specified program cannot be found, a halt occurs with the B-address register containing 00162.
2. Mount the required tapes. If the object program is executed directly after compilation, change the assignment of logical drive 2 (the output object program tape) to logical drive 0.
3. Press the BOOTSTRAP button once.
4. Press the RUN button twice.

Loading from Cards

1. Insert the object deck; when a punched deck is requested, the compiler produces a completely self-loading deck.
2. Press the BOOTSTRAP button once.
3. Press the RUN button twice.

SECTION IX

COMPILATION LISTINGS AND DIAGNOSTICS

Certain information is generated by COBOL Compiler B as a debugging aid and documentation of the source program. This output appears in the following order:

1. Source program listing and embedded diagnostics.
2. Object code memory map.

SOURCE PROGRAM LISTING AND EMBEDDED DIAGNOSTICS

The user's source program, complete with sequence numbers (if specified), is provided. Diagnostics, in the form of single-character keys, are embedded within the source-program listing. The diagnostic keys and their meanings are summarized in Table 9-1. The following printout shows an example of a diagnostic:

```
01510       MOVE ZEROES TO CTR.
                 1             S
```

In most instances, fatal diagnostics do not halt compilation immediately. However, if the compilation is halted by a fatal diagnostic, the programmer should correct all fatal errors in his program before compiling again.

The following diagnostics are generated after compilation and appear at the end of the source program listing:

"paragraph-name" IS NOT DEFINED.
"paragraph-name" IS NOT ALTERABLE.
TOO MANY FILES SELECTED. (generated when seventh SELECT clause is detected)

The above diagnostics are fatal.

OBJECT CODE MEMORY MAP

Subroutine Information

The compiler supplies important information regarding the input/output and other special subroutines generated by the compiler. Following the subroutine name is a listing of the actual memory locations, load parameters, and the encoded instruction. The following example shows this information.

Table 9-1. Summary of Diagnostic Keys

Key	Meaning
1	This word is syntactically incorrect. (Fatal)
2	Column 7 is not blank. (Fatal)
3	Line number sequence error. (Warning)
4	Illegal character in word. (Fatal)
5	Not a COBOL word. (Fatal)
6	More than 30 characters in a word. (Fatal)
7	Either the period is missing at the end of a description of there is a syntax error. (Fatal)
8	This file was not selected in the ENVIRONMENT DIVISION. (Fatal)
10	Only one level of OCCURS is permitted within a hierarchy. (Fatal)
11	An item requiring subscripting cannot be redefined. (Fatal)
12	Redefinition area is bigger than redefined area; layout has been adjusted. (Warning)
13	Previous PICTURE item must be elementary and cannot have a subordinate item. (Fatal)
14	The edited PICTURE was modified to conform to the Series 200 edit instructions. (Warning)
15	The edited PICTURE contained an illegal combination of characters. (Fatal)
17	This record contains more than 4096 characters. (Fatal)
18	Two records in a file have been described with different sizes; padding to the maximum size is assumed. (Warning)
19	This elementary item does not have a PICTURE clause. (Fatal)
C	Illegal character or combination of characters in this PICTURE. (Fatal)
D	Illegal duplication of previous name. (Fatal)
E	ENVIRONMENT DIVISION header missing. (Fatal)
L	FD must contain a LABEL RECORDS clause. (Fatal)
M	Peripheral file cannot have standard labels. (Fatal)
O	A maximum of 30 OCCURS clauses is permitted. (Fatal)
P	Period missing after previous word. (Warning)
Q	This program name is more than 6 characters long. The name will be truncated. This only applies to an all-numeric program-name. (Warning)
R	OCCURS clause is illegal for a record in the FILE SECTION. (Fatal)
S	All words between last diagnosed word and this word have been skipped. (Fatal)
W	VALUE IS clause is legal only for elementary unredefined items. (Fatal)

```
FILE SUBROUTINE FOR FILE-1

000000    60 000310
000310    24 43000412
000314    27 14017100000426
000323    24 65020102
000327    24 65017214
000333    24 43000412
000337    24 65020471
000343    24 65017214
000347    24 43000412
000353    24 65017224
000357    30 5500035300045577
```

Data Division Information

The next printout section lists the DATA DIVISION record description entries, WORKING-STORAGE entries, their respective address assignments, and interspersed initial values. The following example indicates this information.

```
     DATA DIVISION ADDRESS ASSIGNMENTS
     (WITH INTERSPERSED INITIAL VALUES)

         NAME         P LEFT    RIGHT

REC-2                   001052  001223
   TAG2A              W 001052  001066
   TAG2B              W 001136  001173
   TAG2C                001174  001223
      TAG2D           W 001174  001217
      TAG2E           W 001220  001223

000276    60 000276
000275    32 653145236443644441515
```

Punctuation Table

A punctuation table is provided to indicate the punctuation of the file and WORKING-STORAGE areas set during compilation. The following is an example of a punctuation table.

```
 PUNCTUATION

000455    60 000456
000455    63
000471    60 000472
000471    63
000527    60 000530
000527    63
```

Index Register Table

An index register table is included to indicate the initial values of the index registers as set by the compiler. An example follows.

```
                INDEX REGISTERS

               012333    60 000002
               000002    24 02724200
               000002    60 000006
               000006    23 017162
               000006    60 000012
               000012    23 000000
               000012    60 000016
               000016    23 000000
```

PROCEDURE DIVISION Information

The final printout information lists, by paragraph name, absolute memory locations, load parameters, and encoded instructions for the complete PROCEDURE DIVISION. A C indicates the beginning of a clause. The following example shows a paragraph and related code.

```
                PROCEDURE DIVISION



                    BG

               021411 C 24 65021157
               021415   23 012355
               021420   23 012332
               021420   64
               021423 C 24 65021157
               021427   23 012355
               021432   23 012332
               021432   64
               021435 C 24 65021157
               021441   23 012354
               021444   23 012332
               021444   64
               021447 C 24 65700213
               021453   21 45
               021453   61 021334


                THERE ARE 005567 LOCATIONS REMAINING
```

APPENDIX A
COBOL RESERVED WORD LISTS

This appendix contains two lists of words reserved for COBOL compilers. Programmers should note that these words have particular meanings to the compilers and, therefore, should not be used by the programmer as user-created names.

List #1 is the Honeywell COBOL B Reserved Word List and applies only to programs written for COBOL Compiler B. If the programmer has no concern, either present or future, for upward compatibility, he must avoid using the words in this list.

List #2 is the Honeywell COBOL Reserved Word List and applies to all Series 200 compilers up to and including COBOL H. If there exists a chance that a COBOL B program may, at some later date, be compiled by a larger compiler, the programmer must avoid using the words in this list.

I. COBOL B RESERVED WORD LIST

ACCEPT
ADD
ADDRESS
ADVANCING
ALPHABETIC
ALTER
APPLY
ARE
ASSIGN
AT
BEFORE
BLOCK
BY
CARD-PUNCH
CARD-READER
CHANNEL
CHARACTERS
CLOSE
COBOL
CONFIGURATION
DATA
DISPLAY
DIVIDE
DIVISION
DOUBLE-BUFFER
EDIT-OPTION
END
ENVIRONMENT
EQUAL
ERROR
EXIT
FD
FILE
FILE-CONTROL
FILLER
FROM
GIVING
GO
GREATER
IDENTIFICATION
IF
INPUT
INPUT-OUTPUT
INPUT-TAPE
INTO
I-O-CONTROL
IS
LABEL
LESS
LINES
MEMORY
MODEL-120
MODEL-200
MODEL-1200
MODEL-2200
MOVE
MULTIPLY
NO
NOT
NOTE
NUMERIC
OBJECT-COMPUTER
OBJECT-PROGRAM
OCCURS
OF
OFF
OMITTED
ON
OPEN
OUTPUT
OUTPUT-TAPE
PAGE
PERFORM
PICTURE
PRINTER
PROCEED
PROGRAM-ID
QUOTE
READ
RECORD
RECORDS
REDEFINES
REWIND
ROUNDED

I. COBOL B RESERVED WORD LIST (cont)

RUN
SECTION
SELECT
SENSE-SWITCH
SERIES-200
SINGLE-BUFFER
SIZE
SOURCE-COMPUTER
SPACE
SPECIAL-NAMES
STANDARD
STATUS
STOP
SUBTRACT
TAPE
TAPE-UNIT
THAN
THRU
TIMES
TO
VALUE
WITH
WORKING-STORAGE
WRITE
ZERO

II. HONEYWELL COBOL RESERVED WORD LIST

ABOUT
ACCEPT
ACCEPT-CARD-READER
ACCEPT-CONSOLE
ADD
ADDRESS
ADVANCING
AFTER
ALL
ALPHABETIC
ALPHANUMERIC
ALTER
ALTERNATE
AN
AND
APPLY
ARE
AREA
AREAS
ASCENDING
ASSIGN
AT
AUTHOR
AUX-CHANNEL
AUX-CHANNELS
BCD
BEFORE
BEGINNING
BEGINNING-FILE-LABEL
BEGINNING-TAPE-LABEL
BEGIN/PROG/AT
BIT
BITS
BLANK
BLOCK
BLOCK-COUNT
BLOCK-SIZE
BY
CALL
CARD-PUNCH
CARD-READ
CARD-READER
CARD-READERS
CARD-TAPE-READ
CHANNEL
CHANNELS
CHARACTER
CHARACTERS
CHECK
CLASS
CLOCK-UNITS
CLOSE
COBOL
CODE
COLUMN
COMMON-W-STORAGE
COMP
COMP-1
COMP-2
COMP-3
COMPUTATIONAL
COMPUTATIONAL-1
COMPUTATIONAL-2
COMPUTATIONAL-3
COMPUTE
COMPUTER
CONFIGURATION
CONSOLE-KEYBOARD
CONSOLE-TYPEWRITER
CONSTANT
CONTAINS
CONTINUE/PROG/AT
CONTROL
CONTROLS
COPY
CORRESPONDING
DATA
DATE-COMPILED
DATE-WRITTEN
DECLARATIVES
DEFINE
DELETED:WHEN=RESOLVED
DEPENDING
DESCENDING
DETAIL
DIGIT
DIGITS
DISPLAY
DISPLAY-1
DISPLAY-2
DISPLAY-CONSOLE
DISPLAY-PRINTER
DIVIDE
DIVIDED
DIVISION
DOLLAR
DOWN
DUMP
DUMP-FILE
ELSE
END
ENDING
ENDING-FILE-LABEL
ENDING-TAPE-LABEL
END-OF-FILE
END-OF-TAPE
ENTER
ENVIRONMENT
EQUAL
EQUALS
ERROR
EVERY
EVF-SIGNAL
EXAMINE
EXCEEDS
EXIT
EXPONENTIATED
FD
FILE
FILE-CONTROL
FILE-LABEL
FILLER
FINAL

II. HONEYWELL COBOL RESERVED WORD LIST (cont)

FIRST
FLOAT
FLOATING-POINT
FOOTING
FOR
FORMAT
FORTRAN
FROM
GENERATE
GIVING
GO
GREATER
GROUP
HASHED
HEADING
HIGH-VALUE
HIGH-VALUES
H-1400
H-1800
H-200
H-200-SPECIAL
H-400
H-800
HLT-CTL
HONEYWELL-1400
HONEYWELL-1800
HONEYWELL-200
HONEYWELL-400
HONEYWELL-800
HVF-SIGNAL
ID
IDENTIFICATION
IF
I-O-CONTROL
IN
INCLUDE
INDEX
INDICATE
INITIATE
INPUT
INPUT-OUTPUT
INSTALLATION
INTO
IS
JUSTIFIED
KEY
LABEL
LAST
LEADING
LEAVING
LEFT
LESS
LIBRARY
LIMIT

LIMITS
LINE
LINE-COUNTER
LINES
LOAD
LOADER
LOCATION
LOCK
LOWER-BOUND
LOWER-BOUNDS
LOW-VALUE
LOW-VALUES
MEMORY
MEMORY-DUMP
MEMORY-DUMP-KEY
MINUS
MODE
MODULE
MODULES
MOVE
MULTIPLE
MULTIPLIED
MULTIPLY
MULTIPLY-DIVIDE
NEAC-280
NEGATIVE
NEXT
N-2800
NO
NO-MEMORY-DUMP
NOT
NOTE
NUMBER
NUMERIC
OBJECT-COMPUTER
OBJECT-PROGRAM
OCCURS
OF
OFF
OMITTED
ON
OPEN
OPTIONAL
OR
ORIGINAL:SOURCE LANGUAGE
OTHERWISE
OUTPUT
OVERFLOW
PAGE
PAGE-COUNTER
PAPER-TAPE-PUNCH
PAPER-TAPE-READER
PERFORM
PICTURE

PICTURE
PLACE
PLACES
PLUS
POINT
POSITION
POSITIVE
PREPARED
PRINT
PRINTER
PRINTERS
PRINT-TAPE
PRIORITY
PROCEDURE
PROCEED
PROGRAM-ID
PROTECT
PUNCHES
PUNCH-TAPE
PURGE-DATE
QUOTE
QUOTES
RANGE
READ
READER-PUNCH
READER-PUNCHES
REASSIGNMENT
RECORD
RECORDING
RECORD-COUNT
RECORD-SIZE
RECORDS
REDEFINES
REEL
REEL-NUMBER
RELEASE
REMARKS
RENAMES
RENAMING
REPLACING
REPORT
REPORTING
REPORTS
RERUN
RESERVE
RESET
RETURN
REVERSED
REWIND
RIGHT
ROUNDED
RUN
RWCS
SAME

II. HONEYWELL COBOL RESERVED WORD LIST (cont)

SD
SECTION
SECURITY
SEGMENTATION
SEGMENT-LIMIT
SELECT
SELECTED
SENSE-SWITCH
SENTENCE
SENTINEL
SEQUENCED
SET
SHORT-GAP
SIGN
SIGNED
SIZE
SLAVE-TYPEWRITER
SORT
SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
STANDARD-80
STANDARD-120
STATUS
STOP
SUBTRACT
SUM
SUPERVISOR
SUPPRESS
SYNCHRONIZED
TALLYING
TAPE
TAPE-READ
TAPE-READ-BACKWARDS
TAPE-READ-FORWARD
TAPE-UNIT
TAPE-UNITS
TAPE-WRITE
TEMP-STORAGE
TERMINATE
TEST-PATTERN
THAN
THEN
THROUGH
THRU
TIMES
TO
TRAILING-COUNT
TYPE
UNEQUAL
UNTIL
UPON
UPPER-BOUND
UPPER-BOUNDS
USAGE
USE
USING
VALUES
VARYING
WHEN
WITH
WORDS
WORKING-STORAGE
WRITE
ZERO
ZEROES
ZEROS

SPECIAL SYMBOLS

=
)
(
+
-
*
**
/

APPENDIX B

TABLES

Table I. Character Correspondence Table

COBOL CHARACTER SET	HIGH-SPEED PRINTER SET	KEY PUNCH CHARACTER SET	CARD CODE	MACHINE CODE	NORMAL COLLATION VALUE (OCTAL)
0	0	0	0	000000	00
1	1	1	1	000001	01
2	2	2	2	000010	02
3	3	3	3	000011	03
4	4	4	4	000100	04
5	5	5	5	000101	05
6	6	6	6	000110	06
7	7	7	7	000111	07
8	8	8	8	001000	10
9	9	9	9	001001	11
	'		8, 2	001010	12
=	=	#	8, 3	001011	13
	:	@	8, 4	001100	14
Δ	space	space	blank	001101	15
	¢ non-printing		8, 6	001110	16
	&		8, 7	001111	17
+	+	&	R, 0	010000	20
A	A	A	R, 1	010001	21
B	B	B	R, 2	010010	22
C	C	C	R, 3	010011	23
D	D	D	R, 4	010100	24
E	E	E	R, 5	010101	25
F	F	F	R, 6	010110	26
G	G	G	R, 7	010111	27
H	H	H	R, 8	011000	30
I	I	I	R, 9	011001	31
;	;		R, 8, 2	011010	32
.	.		R, 8, 3	011011	33
)	)		R, 8, 4	011100	34
	%		R, 8, 5	011101	35
	■		R, 8, 6	011110	36
	Δ non-printing		R	011111	37
-	-	-	X, 0	100000	40
J	J	J	X, 1	100001	41
K	K	K	X, 2	100010	42
L	L	L	X, 3	100011	43
M	M	M	X, 4	100100	44
N	N	N	X, 5	100101	45
O	O	O	X, 6	100110	46
P	P	P	X, 7	100111	47
Q	Q	Q	X, 8	101000	50
R	R	R	X, 9	101001	51
	#		X, 8, 2	101010	52
$	$	$	X, 8, 3	101011	53
*	*	*	X, 8, 4	101100	54
"	"		X, 8, 5	101101	55
	↓ non-printing		X, 8, 6	101110	56
	? non-printing		X	101111	57
	◇ non-printing		8, 5	110000	60
	/	/	0, 1	110001	61
S	S	S	0, 2	110010	62
T	T	T	0, 3	110011	63
U	U	U	0, 4	110100	64
V	V	V	0, 5	110101	65
W	W	W	0, 6	110110	66
X	X	X	0, 7	110111	67
Y	Y	Y	0, 8	111000	70
Z	Z	Z	0, 9	111001	71
	@		0, 8, 2	111010	72
,	,	,	0, 8, 3	111011	73
(	(	%	0, 8, 4	111100	74
	C_R		0, 8, 5	111101	75
	□ non-printing		0, 8, 6	111110	76
	⊗ non-printing		0, 8, 7	111111	77

APPENDIX C

THE UPDATE B PROGRAM

The Series 200 Update B program permits more compact storage, easier maintenance, and faster loading of object programs by enabling the user to store them on magnetic tape.

Update B performs either one of two separate functions, depending upon the mode in which it is used. These are:

1. Create Mode - creates a program tape by placing one or more object program card decks or card-image tapes on a self-loading tape (SLT).
2. Update Mode - maintains and updates an existing SLT by producing a new SLT as follows:
    a. Copies programs from the old SLT onto the new SLT.
    b. Deletes old programs by omitting them from the new SLT.
    c. Inserts new programs onto the new SLT from either punched cards or magnetic tape.

INPUT

The input required by Update B depends upon which function is performed. In the create mode, Update B requires as input a control deck and the object programs (on either punched cards or magnetic tape) which are to be placed on the new SLT. In the update mode, the old SLT containing those programs to be copied or deleted is also required.

Old SLT

Required only in the update mode, the old SLT is mounted as logical tape 0. The description of this tape corresponds to that given under "New SLT" below.

Control Deck

The control deck, consisting of a series of director cards which indicate the action to be performed for each input program and each program on the old SLT (update mode only), is read via an on-line card reader. In the create mode, these action directors may be in any sequence; in the update mode,they must be in the same sequence as the programs on the old SLT (with the exception of the Insert directors, which may be interspersed as desired). A description of the format and function of each type of action director is found under "Director Cards" below.

Input Programs

Update B will accept any complete object programs, in the form of punched card decks or card images on tape, which are generated by any of the following Honeywell Series 200 software:

1.	Easycoder Assembler A	(Self-loading format)
2.	Condense A	(Self-loading format)
3.	Easycoder Assembler B	(Self-loading format)
4.	Easycoder Assembler B	(Binary run deck format)
5.	COBOL Compiler B	(Binary run deck format)

In addition, Update B will accept any complete object programs from an old SLT (self-loading format) generated by a previous Update B run or from a Distributor Transaction Tape (self-loading and/or binary run deck format) received from Honeywell.

CARD INPUT

Any object-program card deck which is input to Update B is placed immediately behind its associated action director in the control deck. In the case of an object-program deck generated by COBOL Compiler B in binary run deck (BRD) format, the Loader B program must precede the object program (see Appendix D). Both of these decks are then placed behind the appropriate action director.

TAPE INPUT

An input program tape containing one or more complete object programs to be included on the new SLT can be mounted as logical tape 2. Any action director specifying a program on this tape causes the tape to be searched. If the program is not found, a programmed halt occurs.

Binary Run Deck Format

For each input object program recorded in binary run deck format, Update B generates the following series of self-loading routines. These are written ahead of the object program on the new SLT.

1. Tape Search B Bootstrap record (sets up punctuation and reads in next record).
2. Tape Search B program record. (This single record program locates and initiates the loading of any program on the program tape as called for by a Console Call card. If SENSE switch 4 is set ON, it initiates the loading of the next program on tape without reading a Console Call card.
3. Tape Loader B (loads the BRD-format object program which follows).

Following these is the object program in binary run deck format.

Self-Loading Format

Each object program on tape in self-loading format is handled by Update B as follows:

1. The Tape Search B bootstrap record and the Tape Search B program record are generated ahead of the object program.
2. The object-program bootstrap record is increased in length to 104 characters and the Load card is modified for tape loading.

OUTPUT

New SLT

The new SLT is written on tape drive 1. It is composed of one or more complete object programs, in self-loading or binary run deck format (see Figure C-2). Each object program in binary run deck format is preceded by a Tape Search B bootstrap record, a Tape Search B program record, and the Tape Loader B program. Each object program in self-loading format is preceded by a Tape Search B bootstrap record and a Tape Search B program record; no loader is required since the object program is in self-loading format.

Directory Listing

An optional listing of all programs on the new SLT is produced on an on-line printer.

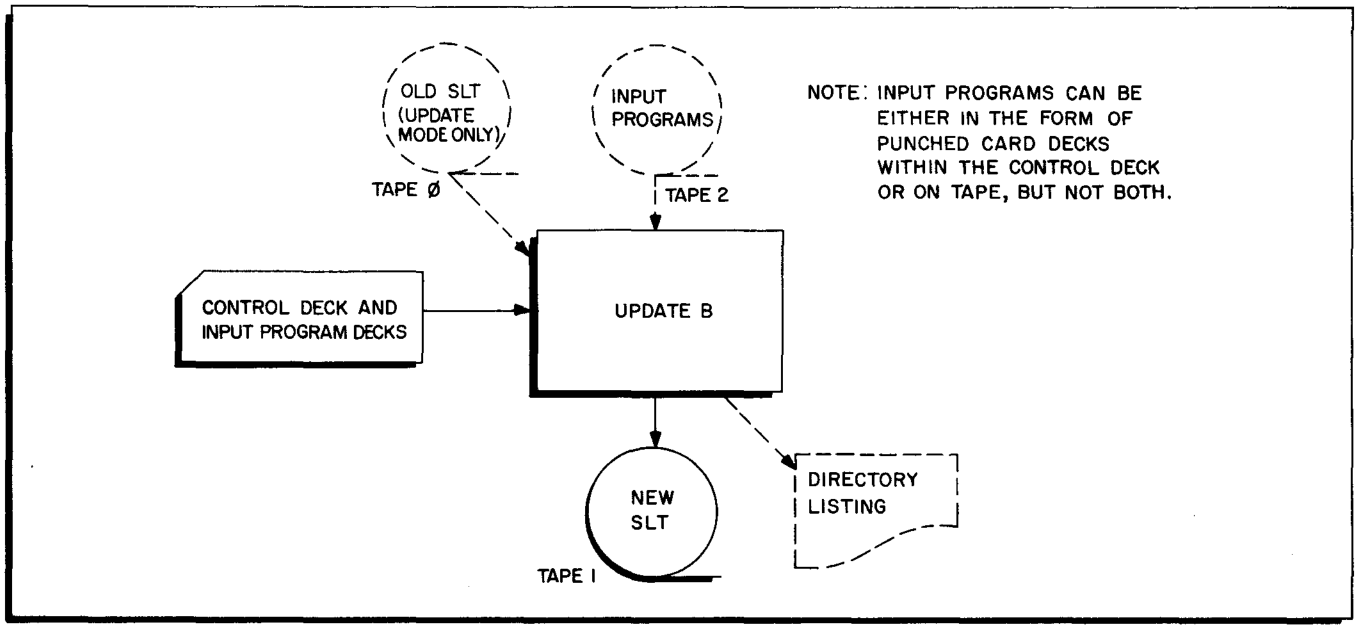

OLD SLT (UPDATE MODE ONLY)
TAPE Ø
INPUT PROGRAMS
TAPE 2
NOTE: INPUT PROGRAMS CAN BE EITHER IN THE FORM OF PUNCHED CARD DECKS WITHIN THE CONTROL DECK OR ON TAPE, BUT NOT BOTH.
CONTROL DECK AND INPUT PROGRAM DECKS
UPDATE B
NEW SLT
TAPE I
DIRECTORY LISTING

Figure C-1. The Update B Program

DIRECTOR CARDS

The director cards which make up the control deck input to Update B consist of three types:

1. Control header card
2. Action directors
3. Trailer card

The control deck, which is always read by an on-line card reader, must begin with a control header card, contain one or more action directors, and terminate with a trailer card.

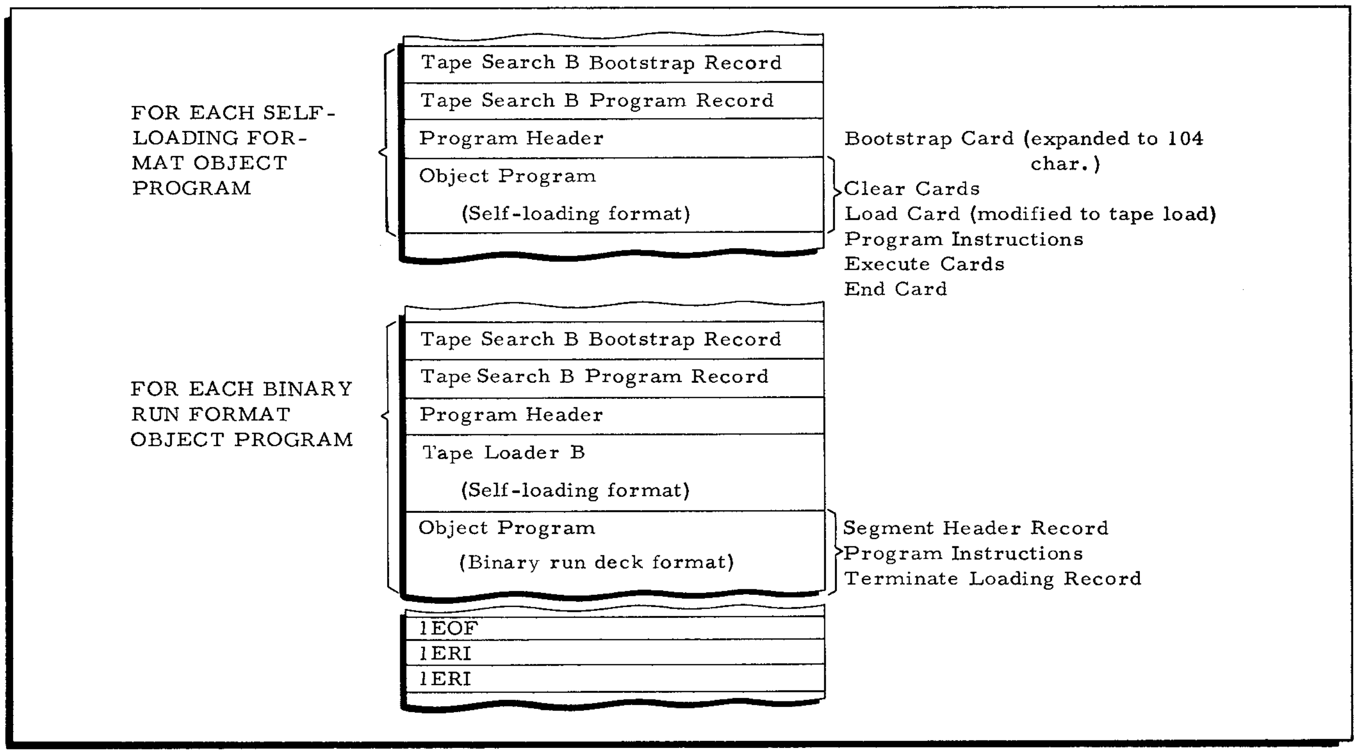

FOR EACH SELF-LOADING FORMAT OBJECT PROGRAM
Tape Search B Bootstrap Record
Tape Search B Program Record
Program Header
Object Program (Self-loading format)
Bootstrap Card (expanded to 104 char.)
Clear Cards
Load Card (modified to tape load)
Program Instructions
Execute Cards
End Card
FOR EACH BINARY RUN FORMAT OBJECT PROGRAM
Tape Search B Bootstrap Record
Tape Search B Program Record
Program Header
Tape Loader B (Self-loading format)
Object Program (Binary run deck format)
Segment Header Record
Program Instructions
Terminate Loading Record
1EOF
1ERI
1ERI

Figure C-2. Self-loading Tape (SLT) Format

Control Header Card

The control header card identifies the beginning of the control deck and initializes Update B to operate in either the create or the update mode. Table C-1 below details the parameters required.

Table C-1. Control Header Card

COLUMN(S)	CONTENTS INTERPRETATION
1-5	Must contain the value 1HDR Δ .
6-14	Blank
15	Specifies whether Update B is to operate in the create or the update mode: C - Indicates that Update B is to operate in the create mode. No old SLT is mounted on tape drive 0 and only insert action directors are permitted in the control deck. U - Indicates that Update B is to operate in the update mode. An old SLT must be mounted on tape drive 0 and all action directors are valid.
16-27	Blanks
28-55	Contains any information, such as the revision number or revision date of the new SLT, to be printed in the header line of the directory listing.
56-80	Blanks

Action Directors

Action director cards are of four types:

1.	Insert -	Directs Update B to insert the named program, which either follows in punched card form or is on the input program tape, onto the new SLT.
2.	Delete -	Directs Update B to bypass copying the named program from the old SLT onto the new SLT.
3.	Copy -	Directs Update B to copy the named program from the old SLT onto the new SLT.
4.	Replace -	Directs Update B to insert the named program, which either follows in punched card form or is on the input program tape, onto the new SLT in place of an identically named program on the old SLT.

INSERT DIRECTOR

In the create mode, Update B requires that there be one Insert director for each program to be written onto the new SLT. The sequence of these directors establishes the order in which these programs will appear on the new SLT. In the update mode, Insert directors may appear anywhere in the control deck and determine at what points the new programs will appear within the order of the programs from the old SLT. The only restriction is that the name of the program to be inserted next cannot be the same as the name of the currently positioned program on the old SLT.

The setting of SENSE switch 2 determines from what source these new programs will be taken. If SENSE switch 2 is OFF, all programs to be inserted are assumed to be on the input program tape (tape drive 2). If SENSE switch 2 is ON, all programs to be inserted must be in punched card form following their respective Insert directors.

The format for the Insert director is described in Table C-2.

Table C-2. Insert Director Card

COLUMN(S)	CONTENTS INTERPRETATION
1-3	INS
4-20	Blanks
21-26	The six-character program name of the object program to be inserted onto the new SLT. This program must either follow this director card (SENSE switch 2 ON) or else be present on the input program tape (SENSE switch 2 OFF).
27-29	Blanks
30-37	Identifying information (creation date, revision number, etc.) - Optional
38-80	Blanks

DELETE DIRECTOR

The Delete director is valid only in the update mode. Since there must be some type of director card present for each program on the old SLT, a Delete director must be supplied for all old-SLT programs which are not otherwise referenced by either a Copy or a Replace director. The absence of a director reference to a program on the old SLT results in a programmed halt.

In effect, the old-SLT program specified in the Delete director card is not copied onto the new SLT. A description of the format for this director is given in Table C-3.

Table C-3. Delete Director Card

COLUMN(S)	CONTENTS INTERPRETATION
1-3	DEL
4-20	Blanks
21-26	The six-character program name of the old-SLT object program which is not to be included on the new SLT.
27-29	Blanks
30-37	Identifying information - Optional
38-80	Blanks

COPY DIRECTOR

The Copy director is valid only in the update mode and causes the named program to be copied from the old SLT onto the new SLT. A Copy director must be included for every old-SLT object program which is not otherwise referenced by a Delete or a Replace director. A description of the format for this director is given in Table C-4.

Table C-4. Copy Director Card

COLUMN(S)	CONTENTS INTERPRETATION
1-3	COP
4-20	Blanks
21-26	The six-character program name of the old-SLT object program which is to be included on the new SLT.
27-29	Blanks
30-37	Identifying information - Optional
38-80	Blanks
NOTE: Since the Update B program itself is normally the first program on the SLT, the first action director in the control deck should be a Copy director containing UPDATB in columns 21-26.	

REPLACE DIRECTOR

The Replace director is valid only in the update mode. It is used whenever an object program on the old SLT is to be replaced on the new SLT by an input program which is either in punched card format following this director (SENSE switch 2 ON) or on the input program tape (SENSE switch 2 OFF). This function is primarily used to replace an old version of an object program with its updated or corrected version. The program name assigned to this new version is the same as that of the original program. In effect, the Replace director causes the deletion of the existing program and the insertion of the new program. The format for this director is described in Table C-5.

Table C-5. Replace Director Card

COLUMN(S)	CONTENTS INTERPRETATION
1-3	REP
4-20	Blanks
21-26	The six-character program name of the old-SLT object program to be replaced on the new SLT by either the program deck which follows (SENSE switch 2 ON) or by an object program of the same name on the input program tape (SENSE switch 2 OFF).
27-29	Blanks
30-37	Identifying information - Optional
38-80	Blanks

Trailer Card

The trailer card is used to signal the end of the input control deck. Upon reading this card, Update B first verifies that the old SLT has also reached the end of file if operating in the update mode. If the end of file has not been reached on the old SLT, the operator has the option of causing the remaining programs to be copied onto the new SLT. Update B terminates the new SLT by writing a "1EOFΔ " record followed by two "1ERIΔ " records. The format for the trailer card is shown in Table C-6.

Table C-6. Trailer Card

COLUMN(S)	CONTENTS INTERPRETATION
1-5	1EOFΔ
6-80	Blanks

OPERATING PROCEDURES

The Update B program can be run on any Series 200 computer having the following minimum equipment configuration:

1. 8,192 characters of main memory.
2. Advanced programming instructions.
3. Tape drives:

Create Mode	-	one tape drive if the input programs are on punched cards; two tape drives if the input programs are on tape.
Update Mode	-	two tape drives if the input programs are on punched cards; three tape drives if the input programs are on tape.

4. One card reader.
5. One printer if a directory listing is desired.

Setup and Loading Procedures for Update B

1. If running in the update mode or if loading the Update B program from tape, mount the old SLT on tape drive 0 and place it in PROTECT status.
2. Mount an available tape for the output SLT on tape drive 1 and place it in PERMIT status.
3. Mount the tape reel containing the input programs (if they are on tape) on tape drive 2 and place it in PROTECT status.
4. Initialize the printer if a directory listing is desired.
5. Press the INITIALIZE button on the central processor control panel.
6. Depending upon the method of loading the Update B program, follow the appropriate procedure below.

LOADING UPDATE B FROM CARDS

1. Place in the card reader the Update B binary run deck followed by the control deck. No blank cards are permitted between the two decks. The input programs, if on cards, must follow the associated action directors.
2. If the input programs are on punched cards, set SENSE switch 2 ON.
3. If a directory listing is desired, set SENSE switch 4 OFF.
4. Set the ADDRESS buttons to 0000_8. Set the CONTENTS buttons to the octal address assignment of the card reader (normally 41_8).
5. Press BOOTSTRAP.
6. Press RUN.

LOADING UPDATE B FROM TAPE WITH A CONSOLE CALL CARD

1. Place in the card reader a Console Call card for Update B (UPDATB* in columns 1-7, blanks in the remaining columns) followed by the control deck. The input programs, if on cards, must follow the associated action directors.
2. Set the ADDRESS buttons to 0000_8. Set the CONTENTS buttons to the octal address assignment of the tape drive containing the old SPT (normally 40_8).

3. Press the BOOTSTRAP button.
4. Press RUN.
5. At the first programmed halt (sequence register should contain 0032_8). set SENSE switch 2 ON if the input programs are on punched cards and set SENSE switch 4 OFF.
6. Press the RUN button to initialize the search for Update B.
7. While the search is being made, set SENSE switch 4 ON if a directory listing is not desired.

LOADING UPDATE B FROM TAPE WITHOUT A CONSOLE CALL CARD

The Update B program must be the first program on the old SLT tape.

1. Place the control deck in the card reader. The input programs, if on cards, must follow the associated action directors.
2. Set the ADDRESS buttons to 0000_8. Set the CONTENTS buttons to the octal address assignment of the tape drive containing the old SPT (normally 40_8).
3. Press the BOOTSTRAP button.
4. Press RUN.
5. At the first programmed halt (sequence register should contain 0032_8). set SENSE switch 2 ON if the input programs are on punched cards and set SENSE switch 4 ON (forces the loading of the first program from the old SLT).
6. Press the RUN button to initiate loading.
7. While Update B is being loaded, set SENSE switch 4 OFF if a directory listing is desired.

Update B Programmed Halts

Table C-7 below lists the programmed halts contained in Update B.

NOTE: aaaaa = location of program name from director card.
bbbbb = buffer address

Table C-7. Update B Programmed Halts

A ADDRESS	B ADDRESS	INTERPRETATION	PRESCRIBED ACTION
00000	00001	Card reader not operable.	Initialize card reader; press RUN.
bbbbb	00010	Read error on old SLT.	Press RUN to. try again.
bbbbb	00012	Read error on program input.	Press RUN to. try again
bbbbb	00021	Write error on new SLT.	Press RUN to. try again
00000	00031	New SLT has reached end of tape.	Press RUN to properly terminate new SLT. Last program is incomplete.

Table C-7 (cont). Update B Programmed Halts

A ADDRESS	B ADDRESS	INTERPRETATION	PRESCRIBED ACTION
bbbbb	00111	Card read error.	Run out cards. Check and re-feed. Last card out. Press RUN.
00000	00200	Printer not operable.	Initialize printer. Press RUN.
bbbbb	00220	Printer error.	Mark line in error. Press RUN to reprint line.
00000	04001	1HDR card missing.	Correct control deck. Press RUN.
00000	04002	Illegal director card, or control deck is out of order.	Correct control deck. Rerun from the beginning.
00000	04003	First card of program is not a Bootstrap card.	Correct program input. Rerun from the beginning.
00000	04004	Card(s) between Bootstrap and Load card are not Clear cards.	Correct program input. Rerun from the beginning.
aaaaa	04005	Named program is not on the input program tape.	Correct control deck. Rerun from the beginning.
00000	04006	Control deck is at 1EOF. More information remains on the old SLT.	Press RUN to continue copy of all remaining programs.
77777	04007	End of run.	Remove output.

LOADING PROGRAMS FROM THE SLT

Program searching and loader initialization are under control of the Tape Search B program, which precedes each object program on the SLT. A call to search for and load either a specific program or the next sequential program can originate from either the operator (manual operation) or another program (automatic operation).

The Tape Search B program exists on the SLT as two records:

1. The Tape Search B bootstrap record, which sets punctuation and loads the Search program record which follows.
2. The Tape Search B program record, which performs the actual search and loader initialization.

The Tape Search B program occupies memory locations $026\text{-}199_{10}$ ($032\text{-}307_8$). In addition, the bootstrapping of the Search B program destroys the previous contents of locations $00\text{-}25_{10}$ ($00\text{-}31_8$).

Operating Characteristics

After the Search program has been bootstrapped, a programmed halt occurs. At this point, the operator can set SENSE switch 4 ON and press the RUN button to cause the loading of the next sequential program on tape, or he can set SENSE switch 4 OFF, insert a Console Call card in the card reader, and press the RUN button to initiate a search for the program specified in the call card. If the search is successful, the program is loaded and started. If the search is unsuccessful (i.e., the end of file is encountered before locating the named program), the program tape is rewound and a programmed halt occurs. A second search from the beginning of the SLT can be initiated by pressing the RUN button

The actual operational steps for loading are as follows:

1. The SLT is mounted on tape drive 0 and placed in PROTECT status. The tape must be either rewound or positioned directly before a Tape Search B bootstrap record following the loading of the previous run. (The SLT is always thus positioned after loading any program except the last.)
2. Press the INITIALIZE button on the central processor control panel.
3. Set the CONTENTS buttons to the octal address assignment for the SLT tape drive (normally 40_8).
4. Set the ADDRESS buttons to 0000_8.
5. Press the BOOTSTRAP button.
6. At the first programmed halt (sequence register contains 0032_8), set up for the desired type of search and load:
    a. Search and Load by Console Call card:
        1) Place the Console Call card (see format, Table D-1) in the card reader and initialize.
        2) Set SENSE switch 4 OFF.
        3) Press the RUN button to read the call card and locate the named program.
    b. Load the Next Sequential Program:
        1) Set SENSE switch 4 ON.
        2) Press RUN to load the next program on the SLT.

TAPE SEARCH B PROGRAMMED HALTS

The programmed halts for Tape Search B are given in Table C-8.

Table C-8. Tape Search B Programmed Halts

SEQUENCE REGISTER	INTERPRETATION	PRESCRIBED ACTION
00032	Tape Search B has been bootstrapped and is ready to search for and load the next program.	1. Console Call Card Mode a. Ready call card in card reader. b. Initialize card reader. c. Set SENSE switch 4 OFF. d. Press RUN. 2. Load Next Program Mode a. Set SENSE switch 4 ON. b. Press RUN.
00043	Column 7 of the Console Call card does not contain an asterisk (*).	Correct the call card, replace in card reader, and press RUN.
00065	Called program has not been found.	SLT has been rewound. Press RUN to search again from beginning of tape.
00121	Tape read error while searching.	Press RUN to accept the record and continue the search.

APPENDIX D

LOADER B

Easytab Utility Programs and COBOL B programs written and compiled by the user can be loaded from either a binary run deck (BRD) or a binary run deck format tape under the operation of Loader B. In addition, such programs can be placed on a self-loading program tape (SLT) through the use of Update B, which is covered in Appendix A.

EQUIPMENT REQUIREMENTS FOR LOADER B

1. A Series 200 central processor. Approximately 510 memory locations are required by Loader B. These include locations $025\text{-}189_{10}$ $(031\text{-}275_8)$ plus the last 345 locations of the highest bank in memory. Loader B also uses index registers X5 and X6.
2. Advanced programming instructions.
3. One card reader or one tape drive.

PROGRAM LOADING

Loading from Cards

1. Place the binary run deck, followed by any required parameter card and/or input data deck, into the card reader. Initialize the card reader.
2. Press the INITIALIZE button on the central processor control panel.
3. Set the CONTENTS buttons to the octal address assignment of the card reader (normally 41_8). Set the Address buttons to 0000_8.
4. Press the BOOTSTRAP button. This causes the Bootstrap card (first card of the binary run program deck) to be read into memory starting at location 0.
5. Press the RUN button. This causes the instructions on the Boostrap card to be executed and results in the subsequent loading of Loader B into memory.
6. The binary run deck following the loader is read and loaded into memory. If SENSE switch 1 is ON, a programmed halt occurs after loading. Press the RUN button to continue. This halt does not occur for the Sort B program.

Loading from Tape

1. Punch a Console Call card containing the name of the program on the binary run deck format tape which the loader is to search for and load. The format for this card is given below.

Table D-1. Console Call Card Format

COLUMNS	CONTENTS
1-6	Program name
7	Asterisk (*)
8-80	Blanks

2. Mount the program tape on tape drive 0 and place it in PROTECT status.
3. Place the Console Call card, followed by any other input cards (parameter card, input deck, etc.), in the card reader. Initialize the card reader.
4. Press the INITIALIZE button on the central processor control panel.
5. Set the CONTENTS buttons to designate the octal address assignment of the program tape (normally 40_8). Set the ADDRESS buttons to 0000_8.
6. Press the BOOTSTRAP button. The first record (bootstrap record) is read from the program tape into memory starting at location 0.
7. Press the RUN button. This causes the execution of the instructions in the bootstrap record.
8. A programmed halt occurs with the sequence register set to 32_8.
9. Press the RUN button.
10. The binary run deck card images following the loader on the program tape are read and loaded into memory. If SENSE switch 1 is ON, a programmed halt occurs after loading. Press the RUN button to continue.

LOADER B PROGRAMMED HALTS

Table D-2 shows the programmed halts contained in the Loader B program.

Table D-2. Programmed Halts for Loader B

A ADDRESS	B ADDRESS	INTERPRETATION	PRESCRIBED ACTION
xxxxx	00104	Tape version only. Program specified in the Console Call Card cannot be found on the program tape.	Recheck the Console Call Card and the program tape mounted on tape drive 0. Correct and restart from the beginning.
xxxxx	17002	Current program has reached end of job and control has been returned to the loader.	Set up, initialize, and bootstrap next program to be executed.
xxxxx	xpp1d	Uncorrectable read error.	Tape Loading: Depress RUN button to retry. Card Loading: Refeed card in error and depress RUN button to reread.
xxxxx	x4011	Tape version only. The current program has called another program or program segment which cannot be found.	Program tape has been rewound. Correct program name in memory, (locations $145\text{-}152_8$), if possible, and press RUN.
xxxxx	14000	This halt occurs after the program has been loaded if SENSE switch 1 is ON.	Perform any actions requested by the programmer and press RUN to begin execution.

NOTE: pp = octal address assignment of loading device. x = contents unspecified.
d = 0 if loading from tape; 1 if loading from cards.

COMPUTER-GENERATED INDEX

HONEYWELL EDP TECHNICAL PUBLICATIONS USERS' REMARKS FORM

TITLE: SERIES 200 COBOL COMPILER B SOFTWARE MANUAL

DATED: APRIL 25, 1966

FILE NO: 123.1205.000B.0-292

ERRORS NOTED:

Fold

SUGGESTIONS FOR IMPROVEMENT:

Fold

FROM: NAME ______________________ DATE ____________

COMPANY ______________________

TITLE ______________________

ADDRESS ______________________

____

Cut Along Line

FIRST CLASS
PERMIT NO. 39531
WELLESLEY HILLS
MASS.

BUSINESS REPLY MAIL

No postage stamp necessary if mailed in the United States

POSTAGE WILL BE PAID BY

HONEYWELL
ELECTRONIC DATA PROCESSING DIVISION
60 WALNUT STREET
WELLESLEY HILLS, MASS. 02181

ATT'N: TECHNICAL COMMUNICATIONS DEPARTMENT

Cut [illegible] Line

Honeywell
ELECTRONIC DATA PROCESSING