HONEYWELL EDP

SOFTWARE MANUAL

SERIES 200

COBOL COMPILERS D & H
ADDENDUM #2

GENERAL SYSTEM: SERIES 200/OPERATING SYSTEM - MOD 1

SUBJECT: Additions and Corrections to the COBOL Compilers D & H Manual (Order No. 065), dated January 4, 1966. In particular, additions have been made to Section IV to reflect extensions to the ENVIRONMENT DIVISION entries for COBOL Compiler H. These extensions provide the ability to use the additional read/write channels and peripheral addresses available on the larger Series 200 central processors.

SPECIAL INSTRUCTIONS: Delete pages IX-3 through IX-9 from the present manual. The additions, deletions, and corrections of this addendum will be reflected in the next edition of the manual.

NOTE: The user can signal the updating of this manual with Addendum No. 2 by inserting this cover immediately after the Addendum No. 1 cover.

DATE: October 31, 1966

FILE NO.: 123.1205.001D.3-065*

9278
10.1066
Printed in U.S.A.

*Underscoring denotes Order Number.

HONEYWELL EDP

SOFTWARE MANUAL

SERIES 200

COBOL COMPILERS D & H

GENERAL SYSTEM: SERIES 200 OPERATING SYSTEM — MOD 1

SUBJECT: Programming and Operating procedures for COBOL Compilers D and H.

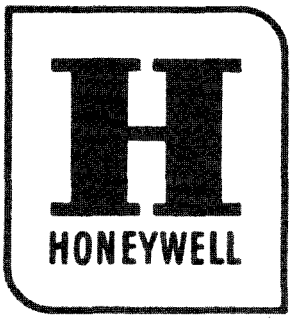
SPECIAL INSTRUCTIONS: This manual completely supersedes COBOL Compiler D Reference Manual (File Number 123.1205.001D.2-065) and COBOL Compiler D Reference Handbook (File Number 124.1205.001D.2-067), both published in November, 1965. It contains all information disseminated in addenda to those publications. This volume also supersedes the software bulletin COBOL Compiler H (File Number 122.1205.001H.00.00). All further editions of this publication will be produced in the same form as this one, i.e., as stapled, loose-leaf pages.

DATE: January 4, 1966

FILE NO.: 123.1205.001D.3-065*

8964
5166
5366
10566
Printed in U.S.A.

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



SIXTH EDITION
First Printing, January, 1966
Second Printing, May, 1966

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."

TABLE OF CONTENTS

TABLE OF CONTENTS (cont)

LIST OF ILLUSTRATIONS

INTRODUCTION

The language of COBOL is essentially independent of any particular computer. Once the programmer has learned the source language and the rules governing it, he can program for any computer for which a COBOL translator or compiler exists. The translation or compiling

In the pages that follow, certain terms are used within special or restricted definitions. For the most part, these terms are common to the data processing field. Two of them, however, should be specially noted since they could give rise to some initial confusion. These words are "block" and "record."

BLOCK is used in its COBOL sense, a physical quantity of computer words or characters recorded in frames on a magnetic tape. A block is preceded and followed by a gap of erased tape.

RECORD is also used in its COBOL sense, a logical unit of information. In reference to data files, a record is the largest unit of logical information. It consists of elementary items which can be combined into group items, which, in turn, can be combined until the record level is reached.

Thus, when speaking of blocks, the reference is to the formatting of records for storage upon the external medium, in this case, magnetic tape. While theoretically a block could contain part of a record, one record, or many records, Series 200 COBOL assumes an integral number of records per block.

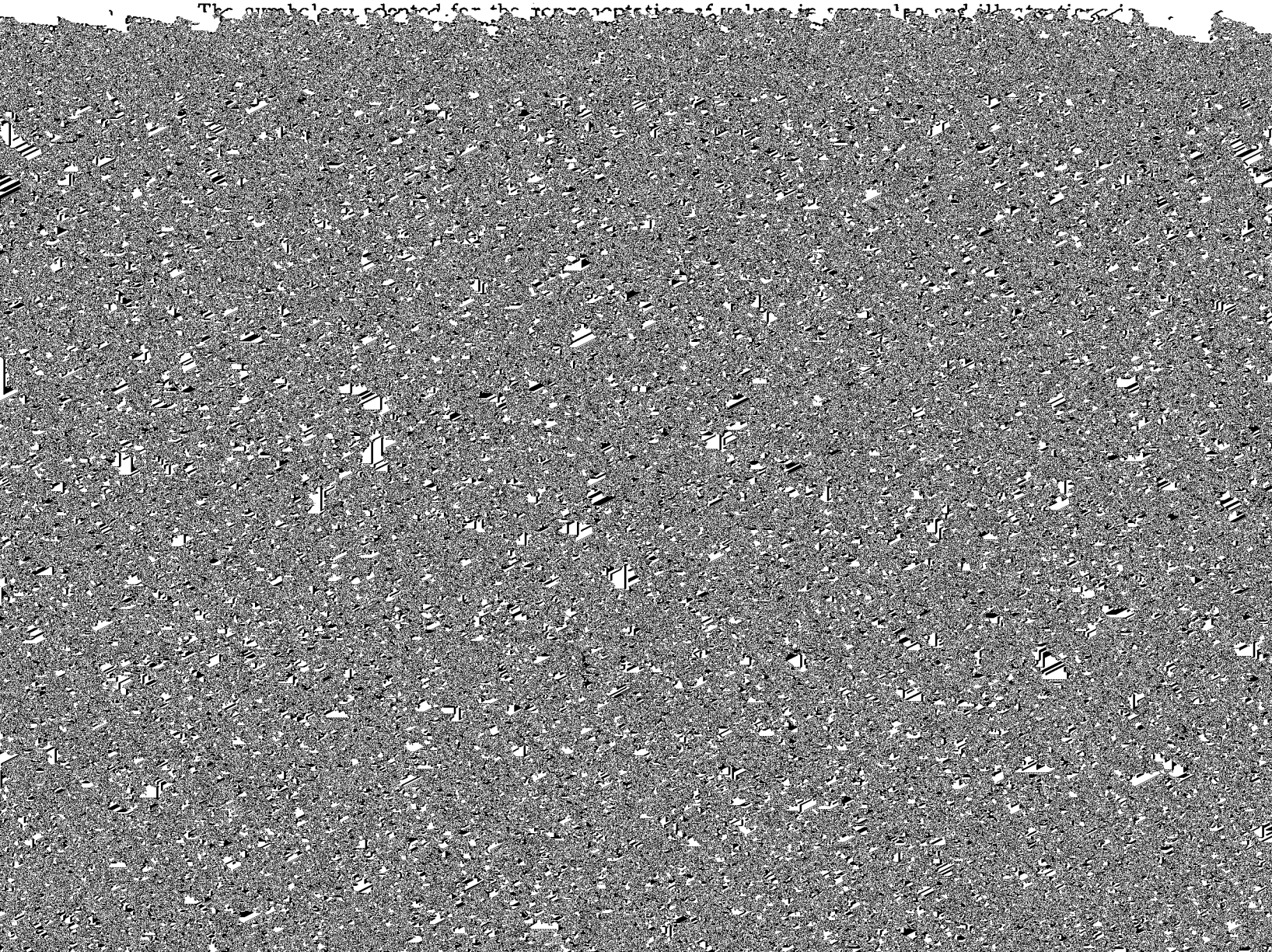


Table 0-1. Equipment Requirements for COBOL Compiler D

Hardware Feature	Type/ Feature No.	At Compile Time		At Object Time	
		Quantity Required	Additional Usable	Quantity Required	Additional Usable

Table 0-2. Equipment Requirements for COBOL Compiler H

Hardware Feature	Type/ Feature No.	At Compile Time		At Object Time	
		Quantity Required	Additional Usable	Quantity Required	Additional Usable
Central processor with control panel and power supply	Any	1	-	1	-
No. of characters of memory	-	32K	Up to max of 32K	32K	Up to max of 262K
Tape control units (1/2" tapes)	Any	1	2	1	Maximum possible
Magnetic tape units (1/2" tapes)	Any	4	0	1	Maximum possible
High-speed printer and control	Any	1	0	0	Maximum possible
Card reader and control	Any	1	0	0	1
Card punch and control	Any	0	1	0	Maximum possible
Paper tape reader and control	-	-	1	0	1
Paper tape punch and control	-	0	1	0	1
Advanced programming instructions	-	1	-	1	-
Editing instructions	013	1	-	1	-
One auxiliary read/write channel	016	0	0	0	1
Extension of print line to 132 characters	031	0	0	0	1

SECTION I

ELEMENTS OF COBOL LANGUAGE

Program Structure

Honeywell COBOL programs consist of four major divisions:

IDENTIFICATION DIVISION - which consists of an identification 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

1. The complete set for Honeywell Series 200 COBOL consists of the following 48 characters:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, (these are the 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 (these are the alphabetic characters)

blank or space (represented by a Δ)

+ (plus sign)

- (hyphen or minus sign)

* (asterisk)

= (equal sign)

$ (dollar sign)

, (comma)

. (period or decimal point)

; (semicolon)

" (quotation mark)

((left parenthesis)

) (right parenthesis)

2. Of the above set, the following 37 characters are used for 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)

3. The characters used for punctuation are:

" (quotation mark)

Δ (blank or space)

((left parenthesis)

) (right parenthesis)

, (comma)

; (semicolon)

. (period)

4. Additional characters, used in editing are:

$ (dollar sign)

* (check protection symbol)

, (comma)

. (actual decimal point)

5. The equal sign (=) is an allowable character in relations.

6. 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) any characters included in the Series 200 high-speed printer character set that are not included in the COBOL set (&, C_r, and ⌀) 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 thirty characters chosen from the thirty-seven characters given in Character Set 2., above. Unless a word is one that names a paragraph or a section in the PROCEDURE DIVISION, it must contain at least one alphabetic character. A paragraph-name can be composed solely of numeric characters. It must be kept in mind that when only numeric characters are used, the name is identified by the characters it contains, not by the numeric value. For example, .0089 is not equivalent as a paragraph-name or a section-name to 89.

A word is ended by a space or by a period, right parenthesis, comma, or semicolon. 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, a comma, or a semicolon is used, it must immediately follow a word (no space can intervene). 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.

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 / ZEROS / ZEROES	Represents the value 0 or the character 0, depending on context.
SPACE / SPACES	Represents one or more blanks or spaces.
QUOTE / QUOTES	Represents one or more of the character " at object time. The data-name QUOTE or QUOTES cannot be used to bound a non-numeric literal.
UPPER-BOUND / UPPER-BOUNDS	Represents the octal value 77, conventionally used as a high delimiter (that is, an upper bound or sentinel) in processing data.
LOWER-BOUND / LOWER-BOUNDS	Represents the octal value 00, conventionally used as a low delimiter (that is, a lower bound or sentinel) in processing data.
HIGH-VALUE / HIGH-VALUES	Represents the octal value 77.
LOW-VALUE / LOW-VALUES	Represents the octal value 00.
ALL any literal	Represents a sequence of the character or characters specified in "any literal." This sequence must be a non-numeric literal or a figurative constant. Although this feature is not a figurative constant in the strict sense of the term, it is included in any discussion concerned with figurative constants. "Any literal" can also be a figurative constant (other than "ALL any literal"). For example, ALL QUOTES is a valid statement. It should be noted, however, that in such an instance ALL is redundant and is used for ease of reading only since just the use of QUOTES is sufficient.

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

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), to describe a data unit, or words which can be used to make a program easier to read. Non-reserved words are those coined by the programmer to create literals, to name units of data, or to express procedure-names.

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

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, a group of successive paragraphs, a section, or a group of successive sections within the PROCEDURE DIVISION. The following discussion is more fully developed in the PROCEDURE DIVISION section of this manual.

1. Statement

   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:

   a. 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.

   b. Conditional statements occur in one of two forms.

      (1) IF condition { statement-1 / NEXT SENTENCE } { OTHERWISE / ELSE } { statement-2 / NEXT SENTENCE }

      where statement-1 and statement-2 can be conditional statements.

      (2) imperative statement followed by a conditional statement

      where imperative statement must not end with a GO TO or a STOP RUN statement.

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

Note: When within a statement there is a series of two or more words equivalent to nouns in the English language, certain words and/or characters can be used as connectives between the words in the series. These connectives are:

,
, AND
, OR
AND
OR

For example, in the statement

ADD data-name-1 data-name-2 data-name-3 data-name-4

connective punctuation could be

(ADD data-name-1, data-name-2, data-name-3, AND data-name-4.)

The use of connectives in a series is optional, unless the words represent conditions and require logical connectives (see the discussion under the IF verb in Section VI, THE PROCEDURE DIVISION).

NOTE: "OR" and ", OR" must not be used as connectives in an arithmetic statement.

2. Sentences

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

a. 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.

b. Conditional Sentences

A conditional sentence consists of a conditional statement terminated by a period. As noted in 1b, above, a conditional statement can itself consist of an imperative statement followed by a conditional statement. For example,

```
IF A IS EQUAL TO B ADD A TO C OTHERWISE ADD B TO C.
ADD A TO B IF B IS GREATER THAN C GO TO PARA-1 OTHER-
WISE PERFORM SECT-2.
```

are conditional sentences.

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.

The following rules apply to the punctuation of sentences.

a. A sentence is terminated by a period.

b. A separator can be used to make sentences easier to read; however, separator use is optional. The allowable separators are

; (semi-colon) (or comma)
THEN

These can be used

(1) between statements;

(2) in a conditional statement between the condition and statement-1 and between statement-1 and OTHERWISE (or ELSE).

c. These separators must not be followed immediately by another such separator.

d. For Series 200 COBOL, a comma can be used interchangeably with a semi-colon; however, such usage may result in source program incompatibility

3. Paragraphs and Sections

A paragraph is a sentence or a group of sequential sentences to which a procedure-name (known as a paragraph-name) is assigned. A section is a paragraph or a group of sequential paragraphs to which a procedure-name (known as a section-name) is assigned. Because paragraphs and sections 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 or section). Paragraphs and sections are executed in their order of appearance within the program, barring any specified transfers of control. (For a detailed description of program control sequence, see 4., Sentence Execution, and 5., Control Relationship Between Procedures, page VI-4 and VI-5. Also, for more information on sections, refer to page VI-5 and VI-6.)

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 (see the OCCURS clause) 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). Tables are often defined so that more than one level of subscripting is required to locate an element within them. A maximum of two levels of subscripting is permitted for tape files, WORKING-STORAGE, and CONSTANT records.

Multi-level subscripts are always written from left to right in the order: major, minor. In this case, the subscripts are shown in a single pair of parentheses, and separated by a comma. For example:

RATE△(REGION,△STATE)
RATE△(3,△STATE)
RATE△(3,△5)

} Commas here are mandatory.

All of the above would reference a particular rate in a two-dimensional table of rates.

A subscript value of "1" denotes the first element of a list, a value of "2", the second element. Thus, a subscript of (1, 2) denotes the second element within the first repeated element

of the table. A table with its main element appearing ten times and its minor element appearing five times within each of the major elements is considered a two-dimensional table. The last element of such a table is referenced by the use of the subscript (10, 5).

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. Reference to a data-name within a table or list must include all subscripts upon which the data-name is dependent. Use of more than, or less than, the correct number of subscripts is considered an error. Some examples of the writing of subscripts are:

```
MOVE RATE (AGE) TO LISTING.
IF HEIGHT (1Ø) 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 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.

The same data-name used as a subscript may be associated with items of differing sizes. Within a given program, the same data-name may appear as one of two subscripts with one item, and as the only subscript with another item.

Subscripting is limited to 32,768 locations in the COBOL D Compiler and 99,999 in the COBOL H Compiler.

Regardless of the above rules, a data-name must not be subscripted under the following conditions:

a. When the data-name is being used as a subscript.

b. When the data-name appears in the defining clauses in the DATA DIVISION.

Qualifying

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

Qualification is performed by appending one or more prepositional phrases, using IN or OF. Since IN and OF are logically equivalent in COBOL, the choice between them when writing

e. A paragraph-name must not be duplicated within the same section. A paragraph-name can only be qualified by a section-name. When it is, the word SECTION must not appear. A paragraph-name need not be qualified when referred to from within the same SECTION. Subscripts and conditional variables, as well as procedure-names and data-names, can be made unique by qualification where necessary or desirable.

f. A data-name cannot be subscripted (see above) when it is being used as a qualifier.

g. A name can be qualified even though it does not need qualification. The use of more names for qualification than are actually required for uniqueness is permitted. If there is more than one combination of qualifiers which ensure uniqueness, then any set can be used.

h. The name of a conditional variable can be used as a qualifier for any of its condition-names.

i. There can be up to 9 levels of qualification.

SECTION II
THE COBOL REFERENCE FORMAT

General

The COBOL source program is written on the Honeywell COBOL Programming Form (1523, Rev. 1) shown on page II-2. From this form, the source program deck for compilation is punched. As a consequence, card column numbers are shown on the form. Certain columns which contain entries constant for each page of coding or for the whole of a source program have been isolated from the main body of the form (PAGE and IDENT) and are found in the upper right- and left-hand portions of each page.

Before discussing the source-program reference format rules within which the programmer uses the programming form, a description of certain recommended standard entries that are, in a way, mechanical and independent of the source program may be useful:

1. Identification of source program coding and deck. An eight-character identifier may be entered in the IDENT field (columns 73 through 80, upper right-hand portion of the coding form). This entry is made on each page of source program coding and punched on each card of the source program deck.

2. Sequence of source coding lines. This entry is divided between two fields, PAGE (columns 1 through 3, upper left-hand portion of the coding form) and SERIAL (columns 4 through 6 in main body of form). The use of PAGE and SERIAL is optional, but their use is recommended to facilitate the updating of source program decks.

   PAGE contains a three-digit number ($000 \leq n \leq 999$). Succeeding page numbers must be sequentially greater than each preceding page number; however, there is no rule as to the increment from one page number to the next. The first page number could be 013, the second could be 090, for example.

   SERIAL consists of two subfields. The first of these (columns 4 and 5) contains a two-digit number ($00 \leq n \leq 99$). Each entry is sequentially higher than the preceding entry for the page. As with PAGE entries, there is no fixed increment associated with the SERIAL entry. The second subfield (column 6, identified by an I) is used primarily for line insertion and contains a one digit number ($0 \leq n \leq 9$). If insertions are not made, this column customarily contains a 0.

   The use of PAGE and SERIAL entries allows for insertion, deletion, and replacement of source coding lines when necessary. When a line is to be inserted, the insertion has the same digits in columns 1 through 5 as the line it is to follow. Column 6 contains a digit sequentially greater than that of the line it is to follow. Since there is no fixed increment between source coding lines, deletion is accomplished by the removal of the line of coding. Replacement is accomplished by removing the line to be replaced and adding the replacement line with the same PAGE and SERIAL number as the line removed.

The remaining entries on the Honeywell COBOL Programming Form are intimately associated with the source program coding. Efforts have been made to provide the programmer with a maximum of flexibility in using the form; however, certain basic format rules must be followed. These rules are given below for each of the divisions that make up a COBOL source program.

Each division begins with a division name followed by a space, the word DIVISION, a period, and a space on the first line. Following lines will contain section and/or paragraph names, followed by the type of entry required. When it is necessary or desired to continue a line of source coding from one form line to another (or to split a word, a numeric literal, or a non-numeric literal from one line to another), the following rules must be followed.

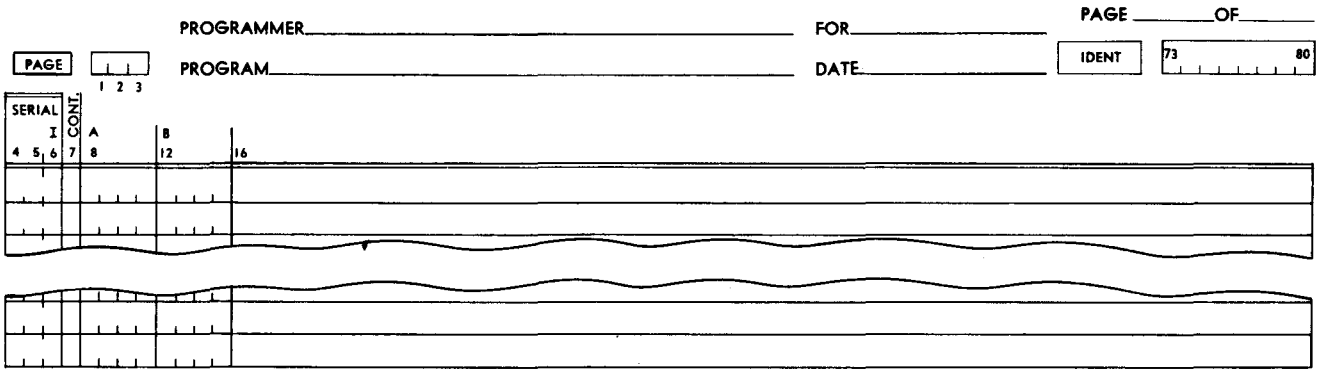
1. To continue a source program line on the succeeding form line, the first continuing word on the second line must begin under position B (column 12). As many spaces as desired can follow the last word on the first line, or the last word of the first line can end at column 72. These spaces are disregarded by Series 200 COBOL compilers.

2. To split a word or numeric literal from one line to another, a hyphen is placed in the CONT. field (column 7) of the second line, and the first continuing character of the word or the literal is placed under position B. As many spaces as desired can occur after the last character of the word or the numeric literal on the first line, or this last character can occur in column 72.

3. To split a non-numeric from one line to another, care must be taken that the last character before the split occurs in column 72. Any spaces following it are regarded as part of the non-numeric literal. A hyphen must be placed in the CONT. field of the second line, and quotation marks must occur under position B. The first character of the literal on the second line occurs in column 13. In the case where the last character of the non-numeric literal occurs in column 72 and only the quotation marks to signal the end of the literal must be entered, the same rules apply (a hyphen in the CONT. field, quotation marks in column 12, and the final quotation marks of the non-numeric literal in column 13).

The above rules for line continuation and word and literal splitting apply to all of the source program divisions.

COBOL PROGRAMMING FORM



Source Program Divisions

1. IDENTIFICATION DIVISION

The IDENTIFICATION DIVISION can consist of up to seven paragraphs. The only required source coding for this division, however, is the division name and the PROGRAM-ID paragraph. As an example of IDENTIFICATION DIVISION reference format, a sample is given below. xxx and yyy represent the PAGE and SERIAL entries, respectively.

As shown in this example, the division-name begins under position A on the first line. The last character of the division name is immediately followed by a period. Nothing else occurs on this line.

COBOL PROGRAMMING FORM

PROGRAMMER _____ FOR _____ PAGE _____ OF _____

PAGE XXX PROGRAM _____ DATE _____ IDENT 73 80

SERIAL	CONT.	A	B
I 4 5 6	7	8	12 16
YYY		IDENTIFICATION DIVISION.	
YYY		PROGRAM-ID. PAYROLL.	
YYY		AUTHOR. JOHN ROE.	
YYY		DATE-WRITTEN. JANUARY 7, 1964.	
YYY		REMARKS. INPUT FROM RUN 4 AND COLLATION WITH RUN 2 OUT-	
YYY	-	PUT TO RUN 6. THIS PROGRAM PROCESSES SALARIED EMPLOYEES	
YYY		ONLY.	

The paragraph-names of the division each begin under position A with the last character of each paragraph-name being immediately followed by a period and at least one space. The text of each paragraph can start anywhere on the same line following the paragraph-name, period, space, or it can begin on the following line under position B. Any paragraph which occupies more than one line must be continued according to the rules for continuation given above.

Specific details concerning the entries to follow the paragraph-names of the IDENTIFICATION DIVISION can be found on page III-1.

2. ENVIRONMENT DIVISION

The reference format for the ENVIRONMENT DIVISION is the same as that of the IDENTIFICATION DIVISION. In addition to fixed paragraph-names, there are also fixed section-names. These section-names, like the division-name, appear as single entries on individual lines. The last character of the section-name is followed by a space, the word SECTION, and a period and a space. The section-names begin under position A. The following is an example of a possible ENVIRONMENT DIVISION format. Specific details as to the entries and information for each section and paragraph can be found in Section IV.

COBOL PROGRAMMING FORM

PROGRAMMER _____ FOR _____ PAGE ____ OF ____

PAGE ___ PROGRAM _____ DATE _____ IDENT 73 80

1 2 3

SERIAL 1 4 5 6	CONT. 7	A 8	B 12 16
Y,Y,Y		ENVIRONMENT DIVISION.	
Y,Y,Y		CONFIGURATION SECTION.	
Y,Y,Y		SOURCE-COMPUTER. COMPUTER-NAME...	
Y,Y,Y		OBJECT-COMPUTER. COMPUTER-NAME...	
Y,Y,Y		SPECIAL-NAMES. HARDWARE-NAME...	
Y,Y,Y		INPUT-OUTPUT SECTION.	
Y,Y,Y		FILE-CONTROL. SELECT FILE-NAME-1 ...	
Y,Y,Y		SELECT FILE-NAME-2... SELECT FILE-NAME-3...	
Y,Y,Y		I-O-CONTROL. APPLY...	

3. DATA DIVISION

The basic unit in the DATA division is the entry. Each entry begins with a level number (see page V-25) followed by at least one space, the name of the data item, and a sequence of independent clauses descriptive of the item. Each clause except the last can be terminated by a semicolon followed by a space. The last clause is terminated by a period and a space. The same format rules apply to the file and record description portions, as well as the Working-Storage and the Constant Sections of the DATA DIVISION.

COBOL PROGRAMMING FORM

PROGRAMMER_____ FOR_____ PAGE _____OF_____

PAGE [ ] 1 2 3 PROGRAM_____ DATE_____ IDENT 73 80

SERIAL 1 4 5 6	CONT 7	A 8	B 12	16
Y,Y,Y		DATA	DIV	ISION.
Y,Y,Y		FILE	SEC	TION.
Y,Y,Y		FD	MAST	ER-PAYROLL; LABEL RECORDS ARE OMITTED; DATA
Y,Y,Y			RECO	RD IS MASTER-PAY.
Y,Y,Y		Ø1	MAST	ER-PAY;
Y,Y,Y			Ø2	BADGE-NUMBER;
Y,Y,Y				PICTURE IS AAAXXX999999.
Y,Y,Y			Ø2	DATE CLASS IS NUMERIC.
Y,Y,Y				Ø3 MONTH; PICTURE IS 99.
Y,Y,Y				Ø3 DAY; PICTURE IS 99.
Y,Y,Y				Ø3 YEAR; PICTURE IS 99.
Y,Y,Y			Ø2	GROSS-PAY; PICTURE IS Ø999V99.

4. PROCEDURE DIVISION

The reference format for the PROCEDURE DIVISION is the same as that of the IDENTIFICATION DIVISION. The first line of the Division consists of its division-name followed by a period, starting under position A. If a section has been designated, the section-name starts under position A, followed by a space, the word SECTION, a space followed by a priority number, when used, a period, and a space.

A paragraph consists of one or more successive sentences, the first of which must be preceded by a paragraph-name. The paragraph-name starts under position A and is followed immediately by a period and a space. The first sentence of the paragraph can begin on the same line as the paragraph name or under position B of the succeeding line. A new paragraph is determined by the appearance of another paragraph-name. Note that a paragraph can consist of a single sentence.

Any sentence which occupies more than one line must be continued under position B on the next line. The rules for splitting words or literals are the same as those given above.

The following is an example of PROCEDURE DIVISION reference format:

COBOL PROGRAMMING FORM

PROGRAMMER _____ FOR _____ PAGE _____ OF _____

PAGE ___ PROGRAM _____ DATE _____ IDENT 73 80

SERIAL I 4 5 6	CONT. 7	A 8	B 12	16
Y Y Y		P R O C	E D U R E DIVISION.	
Y Y Y		C O M P	U T A T I O N S SECTION.	
Y Y Y		U P D A	T E - M A S T E R . MOVE ADJUSTED-PAY TO NET-PAY. ADD	
Y Y Y			G R O S S - PAY TO GROSS - YEAR-TO-DATE. WRITE UP-DATED-	
Y Y Y	-		M A S T E R - PAY. READ MASTER-PAYROLL-RECORD.	
Y Y Y			G O TO INITIALIZATION - ROUTINE.	
Y Y Y		E R R O	R - 1 - ROUTINE.	
Y Y Y			D I S P L A Y "ERROR FOR" TIME-CARD-NUMBER UPON PRINTER.	
Y Y Y			R E A D TIME-CARD ; AT END GO TO CLOSE-OUT.	
Y Y Y			G O TO INITIALIZATION - ROUTINE.	

KEYPUNCHING THE SOURCE PROGRAM

Because 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

A. The source program should be printed (~~if the programmers' handwriting is not legible to the keypunch operator.~~)

B. 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 0.
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 bars 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.

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

Keypunching Conventions

A. The entry in the IDENT. (IDENTIFICATION) space on the upper right of the COBOL Programming Form must be punched in each card of the source program deck in columns 73 through 80.

B. The entry in the PAGE space on the upper left of the form must be punched in columns 1 through 3 of each card punched from the corresponding page. When this entry is blank, the columns are left blank. Normally, the progression is 001, 002, etc. However, this is not required or has no the

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

SECTION III

THE IDENTIFICATION DIVISION

General

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.
PROGRAM-ID. any 1 through 30 characters from the COBOL set.
[DATE-WRITTEN. any literal of 30 characters or less.]
[DATE-COMPILED. any sentence.]
[AUTHOR. any statement or sentence.]
[INSTALLATION. any statement or sentence.]
[SECURITY. any statement or sentence.]
[REMARKS. any statement or sentence.]

Technical Notes

1. IDENTIFICATION DIVISION must be the first entry.
2. The first six characters of the COBOL word following PROGRAM-ID should (but need not) occur on the Program Director Card when the source program is scheduled for a compilation run.
3. The statement or sentence following DATE-COMPILED is replaced at compilation time by the current date, so long as DATE-COMPILED begins in column 8 (position A).
4. Entries other than the Division Header (IDENTIFICATION DIVISION) and PROGRAM-ID are optional. Any statement or sentence following any of them is reproduced on the source program listing.

SECTION IV

THE ENVIRONMENT DIVISION

General

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 over-all specifications of the source and the object computer and is divided into three paragraphs:
   a. SOURCE-COMPUTER.
   b. OBJECT-COMPUTER.
   c. SPECIAL-NAMES.
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 of the Environment Division

CONFIGURATION SECTION

I. CONFIGURATION SECTION

FUNCTION: To define the over-all specifications of the source and object computers.

SECTION FORMAT:

```
CONFIGURATION SECTION.   [ option 1:
                           COPY library-call.

                         option 2:
                           [ H-200
                             HONEYWELL-200
                             H-200-SPECIAL ].                    ]

SOURCE-COMPUTER.

                         [ option 1:
                           COPY library-call.

OBJECT-COMPUTER.         option 2:                                 [, WITH SUPERVISOR CONTROL]
                           { H-200
                             HONEYWELL-200 }                     ]

[, MEMORY SIZE [ integer-1 { CHARACTERS
                             MODULES } ]                         ]
                [ ADDRESS integer-2
                    THRU integer-3                               ]

, RWCS { "1"
         "2"
         "3"
         "1A"
         "4"
         "5"
         "6"
         "6A" } [, { "1"
                     "2"
                     "3"
                     "1A"
                     "4"
                     "5"
                     "6"
                     "6A" } ] ...

[, [integer-4] TAPE-UNIT [S] ]   [, [integer-5] READER-PUNCH [ES] ]

[, [integer-6] PRINTER [S] ]   [ COMMON-W-STORAGE ]   [, AUX-CHANNEL]

[ [, ACCEPT-CONSOLE { m
                      integer-1 } ]
  [, ACCEPT-CARD-READER { m
                          integer-1 } ]   [ DISPLAY-CONSOLE { n
                                                              integer-2 } ]
                                          [ DISPLAY-PRINTER { n
                                                              integer-2 } ] ]

[, { SEGMENT-LIMIT IS integer-7
     NO SEGMENTATION } ]        [NO FILE REASSIGNMENT]

[, ASSIGN OBJECT-PROGRAM TO { TAPE-UNIT
                              READER-PUNCH } ]   [HLT-CTL]   . ]
```

TECHNICAL NOTES:

1. This section must always appear in a source program.
2. Both the SOURCE-COMPUTER and the OBJECT-COMPUTER paragraphs must appear in this section in the format given above.
3. The presence of punctuation marks, except for periods, is optional.
4. The individual paragraphs of this section are described in the following pages.
5. When the READER-PUNCH option is specified, it is assumed that the programmer intends to use a utility routine to punch the object program. Any conflict with SEGMENTATION, either implied or explicit, causes a diagnostic.

SOURCE-COMPUTER

A. SOURCE-COMPUTER

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

FORMAT:

option 1:

SOURCE-COMPUTER. COPY library-call.

option 2:

SOURCE-COMPUTER. { H-200 / HONEYWELL-200 / H-200-SPECIAL } .

TECHNICAL NOTES:

1. The SOURCE-COMPUTER paragraph must appear in a source program.
2. Option 1 is used only when the COBOL library contains the entire SOURCE-COMPUTER paragraph, as shown in option 2.
3. The minimum on-line system upon which Series 200 COBOL compilation can be accomplished is:

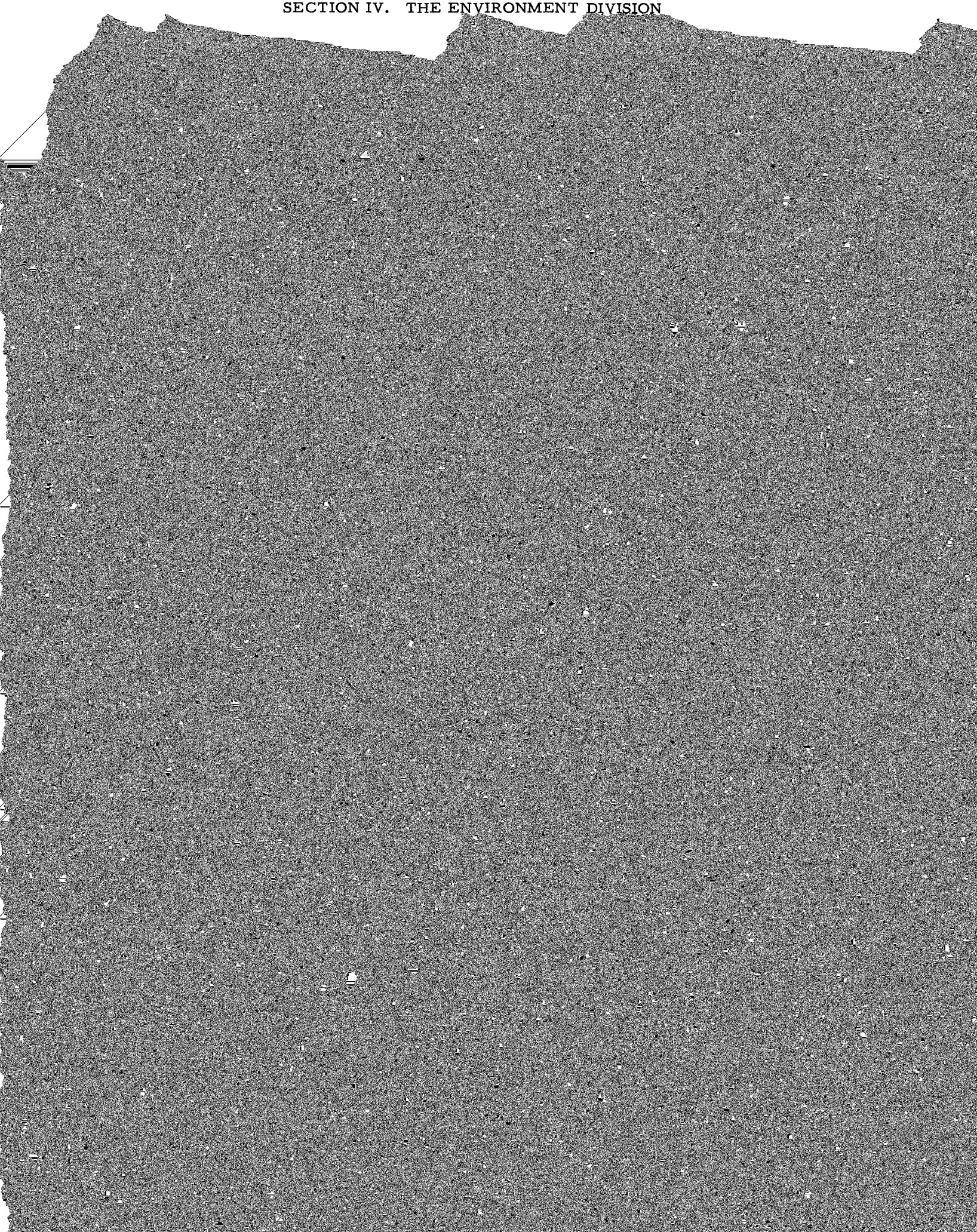
   16,384 characters of high-speed memory.
   4 magnetic tape units.
   1 card reader (on-line or, if an extra tape unit is available, off-line).
   1 printer

4. While the compiler operates on any system that includes the minimum requirements specified in 3., above, no additional components other than high-speed memory are used by the compiler. The presence of additional high-speed memory allows the building of larger internal tables for compiler use.
5. The Series 200 card-reader handles two different character sets. The only difference between the two is in the interpretation of two characters. The interpretation of these characters in source program literals is determined by the clause following SOURCE-COMPUTER.

Character	Card Code	Bit Configuration With H(ONEYWELL)-200	Bit Configuration With H-200-SPECIAL
&	R (12)	011111	010000
	R-0 (12-0)	010000	011111
-	X (11)	101111	100000
	X-0 (11-0)	100000	101111

6. The Type 222 Printer allows the following number of print positions per line at compilation time:

Type Number	Print Positions/Line
222-1	96 (some truncation)
222-2	108
222-3	120 (132 if extended)
222-4	120 (132 if extended)



B. OBJECT-COMPUTER

OBJECT-COMPUTER

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

FORMAT:

option 1:

```
OBJECT-COMPUTER.  COPY library-call.
```

option 2:

```
OBJECT-COMPUTER. { HONEYWELL-200 }  [WITH SUPERVISOR CONTROL]
                 { H-200         }

[, MEMORY SIZE { integer-1 { CHARACTERS } }                                    ]
               { integer-1 { MODULES    } }                                    ]
               { ADDRESS integer-2 THRU integer-3                             ]

[,RWCS { "1"  } [ { "1"  } ... ]
       { "2"  }   { "2"  }
       { "3"  }   { "3"  }
       { "1A" }   { "1A" }
       { "4"  }   { "4"  }
       { "5"  }   { "5"  }
       { "6"  }   { "6"  }
       { "6A" }   { "6A" }

[, [integer-4] TAPE-UNIT [S]] [, [integer-5] READER-PUNCH [ES]]

[, [integer-6] PRINTER [S]]  [HLT-CTL]

[AUX-CHANNEL]  [COMMON-W-STORAGE]

[{,  ACCEPT-CONSOLE     { m         }}]  [{ DISPLAY-CONSOLE { n         }}]
 {,  ACCEPT-CARD-READER { integer-1 }}   { DISPLAY-PRINTER { integer-2 }}

[, { SEGMENT-LIMIT IS integer-7 }]  [NO FILE REASSIGNMENT]
   { NO SEGMENTATION            }

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

TECHNICAL NOTES:

1. The OBJECT-COMPUTER paragraph must appear in every source program.
2. Option 1 is used only when the COBOL library contains the entire OBJECT-COMPUTER entry, as shown in option 2.
3. If the entry consists only of

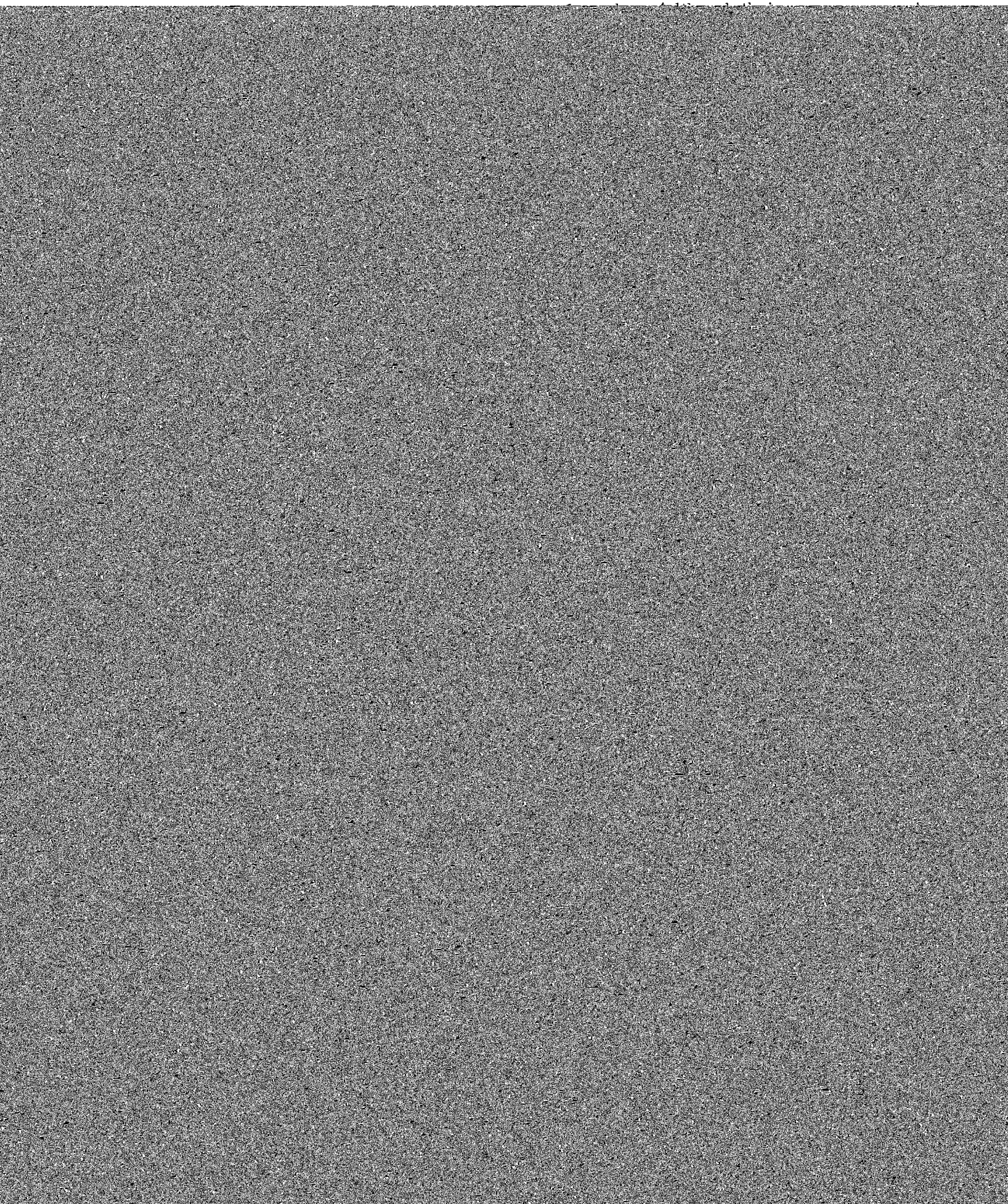
   OBJECT-COMPUTER { H-200 } .
                   { HONEYWELL-200 }

   and no SOURCE-COMPUTER description except H(ONEYWELL)-200 or H-200-SPECIAL, the configuration assumed for the object program to run on is 16384 characters of high-speed memory.
4. The COBOL Compiler D generates 3-character addressing object code.

   The COBOL Compiler H generates 3- or 4-character addressing object code. If, in the

   MEMORY SIZE...

clause, a memory size of less than 32,768 locations (15 bits) is specified, 3-character addressing results. If a memory size of more than 32,768 locations is specified, 4-character addressing results. The highest location



14. The clause

WITH SUPERVISOR CONTROL

is used to determine the processing of all STOP RUN statements. If this clause is present, control is returned to the PLUS loader; otherwise a permanent halt is generated.

15. The object program produced by the compiler is normally written on the binary-run tape in a format loadable by the Tape Loader-Monitor C. However, if it is desired to produce the object program on cards the clause

ASSIGN OBJECT-PROGRAM TO READER-PUNCH

must be specified. For source language programs that contain this clause and have no fatal errors, the compiler punches a card deck. If WITH SUPERVISOR CONTROL was not specified, the deck is completely self-contained and self-loading. It can be bootstrapped into memory at location 317. The loader provided is a minimal one and has no restart, search, own-code, visibility, fixed-start, or halt-after-loading capability. If more flexibility is desired, the

WITH SUPERVISOR CONTROL

clause should be specified. In this instance, the output deck does not contain the minimal loader and the user must supply (or have already in the object-computer memory) the Card/Tape Loader A.

16. The

ASSIGN OBJECT-PROGRAM...

clause implies the NO SEGMENTATION clause; if this clause is not specified, it is assumed present. In order to minimize the memory-overload problems which sometimes arise when NO SEGMENTATION is specified (or implied), integer-2 in the

MEMORY SIZE ADDRESS

clause is permitted to be as low as 400. This is possible since the loader supplied by the compiler is much smaller than the Card/Tape Loader A. Further, the compiler will take advantage of the space factor even if integer-2 is not specified. However, if

WITH SUPERVISOR CONTROL

is specified, the standard requirements on integer-2 remain in effect. Memory overload can be further reduced by using the

NO FILE REASSIGNMENT

clause. In this case, the user loses the ability to reassign files to different devices at object-time, but he frees about 950 locations for use by other code. If no file reassignment is specified, the setting of SENSE Switch 1 to control the reading of the DECD card at object run time is ignored. This option is available in all programs independent of the presence of the

ASSIGN OBJECT-PROGRAM. . .

clause. However, it has a significant effect only in NO SEGMENTATION programs.

Source programs containing the

ASSIGN OBJECT-PROGRAM. . .

clause may be intermixed in any way with other programs in the same batch. Only those programs containing the clause will be punched. Each punched deck will be terminated by three blank cards for easy separation. At the end of a batch compilation, all compiled programs, regardless of whether they were punched or not, are available on the binary-run tape.

17. When the user desires to retain the data-contents of working-storage from the previous program, he may specify the

    COMMON-W-STORAGE

    option. In this case the data is retained and only the punctuation is cleared. The user need not concern himself with memory allocation, because working-storage is always the first memory file allocated.

18. If

    ASSIGN OBJECT-PROGRAM TO READER-PUNCH

    has been specified, COMMON-W-STORAGE is not allowed.

19. The option

    HTL-CTL

    allows the user to eliminate halts at object time. When this option is specified, WITH SUPERVISOR CONTROL is assumed. See Section VII, "Object Messages and Error Halts," for the effects of this option upon the object program.

SPECIAL-NAMES

C. 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:

option 1:

```
[ SPECIAL-NAMES.   COPY library-call. ]
```

option 2:

```
[ SPECIAL-NAMES. [ { PAGE                                          } ]
                 [ { CHANNEL a OF PRINTER { m         } }  IS mnemonic-name-1 ]
                 [                        { integer-1 } ]                   ]

  [ { PAGE                                          } ]
  [ { CHANNEL b OF PRINTER { n         }  IS mnemonic-name-2 ]  . . .
  [                        { integer-2 } ]                   ]

  [ { SENSE-SWITCH c                                  } ]
  [ { EVF-SIGNAL OF PRINTER { m         }  [IS mnemonic-name-3] ]
  [ { HVF-SIGNAL OF PRINTER { integer-1 } ]                     ]

        { , ON STATUS IS condition-name-1  } ]
        { , OFF STATUS IS condition-name-2 } ]

  [ { SENSE-SWITCH d                                  } ]
  [ { EVF-SIGNAL OF PRINTER { m         }  [IS mnemonic-name-4] ]
  [ {                       { integer-2 } ]                     ]
  [ { HVF-SIGNAL OF PRINTER { n         } ]                     ]
  [                         { integer-2 } ]                     ]

        { , ON STATUS IS condition-name-3  } ]  . . .   . ]
        { , OFF STATUS IS condition-name-4 } ]           ]
```

TECHNICAL NOTES

1. The SPECIAL-NAMES paragraph is not required if mnemonic-names and sense-switch-names are not used in the source program PROCEDURE DIVISION.
2. Option 1 is used only when the COBOL library contains the entire SPECIAL-NAMES paragraph (except the paragraph-name) as shown in option 2.

3. The assigned mnemonic-name for PAGE is used to advance the printer page to head-of-form. For the Type 222 Printer, the assigned mnemonic-name for CHANNEL b OF PRINTER m (or integer-1) is used to advance the printer page to a position governed by the appropriate channel of the vertical-format, paper-tape loop. In addition, the assigned condition-name for EVF-SIGNAL and/or HVF-SIGNAL is used to sense the position of the previously-mentioned loop. EVF and HVF refer to channels 2 and 8 respectively.

4. Mnemonic-name-3 is for documentary use only.

5. A mnemonic-name cannot be written in a source program except where formats specifically show that the use of a mnemonic-name is permitted.

6. Whenever a SENSE-SWITCH, EVF-SIGNAL, or HVF-SIGNAL entry is made, at least one of the three bracketed entries which follow in the above format must also be used. In a SENSE-SWITCH entry, c (or d) indicates the number of a switch, 1 through 4. In a CHANNEL entry, a (or b) indicates the channel number, 1 through 8.

7. COBOL D allows the peripheral address of the PRINTER to be indicated by an alphabetic character (m or n) in the range of A through H. COBOL H also allows the address to be specified as an octal number (integer-1, -2); e.g., 03.

II. 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.   { Option 1:
                    COPY library-call.
                  { Option 2:
                    SELECT [OPTIONAL]  file-name-1  [RENAMING file-name 2]

      ASSIGN TO   { TAPE-UNIT  { mn        } [,  TAPE-UNIT { Pq        } ]
                  {            { integer-1 }                { integer-2 }
                  { CARD-READER { m       }
                  {             { integer-3 }
                  { CARD-PUNCH  { m       }
                  {             { integer-3 }
                  { PRINTER     { m       }
                  {             { integer-3 }

    [FOR MULTIPLE REEL] [, RESERVE NO ALTERNATE { AREA  } ]  .  [ SELECT... . ] ]
                                                { AREAS }

[ I-O-CONTROL     { Option 1:
                    COPY library-call.
                  { Option 2: [APPLY { H-200-SPECIAL   } ON file-name-1 [, file-name-2...]]
                                     { TRAILING-COUNT  }
                                     { SHORT-GAP       }

    [; SAME AREA FOR file-name-5, file-name-6 [, file-name-7... ]]

    [; MULTIPLE FILE TAPE CONTAINS file-name-8, file-name-9 [, file-name-10 ... ]]

    [; RERUN [ON file-name-11]  EVERY { END OF REEL      } OF file-name-12]  .  } ]
                                      { integer-1 RECORDS }
```

TECHNICAL NOTES:

1. This section must appear in the source program whenever there are input or output files.
2. When this section appears in the source program, the paragraphs must appear in the order given above.
3. The presence of punctuation marks, except for periods, is optional.
4. The individual paragraphs of this section are described in the following pages.

A. FILE-CONTROL

FILE-CONTROL

FUNCTION: To name each file, identify its medium, and allow particular hardware assignments. To specify alternate input-output areas.

FORMAT:

option 1:

```
[FILE-CONTROL,  COPY library-call. ]
```

option 2:

```
[FILE-CONTROL,  SELECT [OPTIONAL] file-name-1  [RENAMING file-name 2]
                ASSIGN TO { TAPE-UNIT {mn / integer-1} [, TAPE-UNIT {pq / integer-2}...]
                          | CARD-READER {m / integer-3}
                          | CARD-PUNCH {m / integer-3}
                          | PRINTER {m / integer-3} }
   [FOR MULTIPLE REEL] [, RESERVE NO ALTERNATE {AREA / AREAS}] . [SELECT ... .]]
```

TECHNICAL NOTES:

1. Option 1 is used when the COBOL library contains the entire FILE-CONTROL paragraph (except the paragraph-name) as shown in option 2.
2. The beginning of the information for each file-name-1 must be identified by the key word SELECT.
3. Each file which is used in the program must be used as file-name-1 in a FILE-CONTROL paragraph, following the key word SELECT, once and only once. The name of each SELECTed file (i. e., file-name-1) must be unique within a program. Each file must have a File Description in the DATA DIVISION of the source program.
4. OPTIONAL is required for input files which are not necessarily present each time the object program is run. See Section VII for operating instructions.
5. The RENAMING option is a Language Preprocessor element. See Section XI for a complete discussion.
6. mn and pq are equivalent to the standard Honeywell trunk and tape assignment notation of AA through HH.
7. I/O trunk assignment is indicated by m; i. e., m = A through H.
8. The variable letters in the ASSIGN and ACCEPT/DISPLAY statements are used by the compiler to specify peripheral control unit addresses in the following manner:
   a. Magnetic tapes
      (1) The rightmost two bits of the letter become the rightmost two bits of the C2 variant in the Peripheral Data Transfer instruction (PDT) or the Peripheral Control and Branch instruction (PCB).

(2) The input/output bit (leftmost bit of the C2 variant) is generated from the OPEN INPUT or OPEN OUTPUT statement.

Letters (m, p)	PCU	Letters (n, q)	Drive No.
A(21)	n1	A	1
B(22)	n2	B	2
C(23)	n3	C	3
D(24)	n0	D	4
E(25)	n1	E	5
F(26)	n2	F	6
G(27)	n3	G	7
H(30)	n0	H	8

(If OPEN INPUT is specified, n equals 4.
If OPEN OUTPUT is specified, n equals 0.)

(3) COBOL H, which can handle up to 32 peripheral address assignments, also allows the user to specify these addresses and the logical tape drive by a three-digit octal number (integer-1, 2). The first two octal digits specify the peripheral address; the last digit specifies the logical tape drive number. Thus, TAPE-UNIT 403 is equivalent to TAPE-UNIT DC.

b. Peripheral and accept/display devices

(1) The rightmost two bits of the letter become the rightmost two bits of the C2 variant in the Peripheral Data Transfer instruction (PDT) or the Peripheral Control and Branch instruction (PCB).

(2) The four bit of the letter becomes the input/output bit (the leftmost bit of the C2 variant.

(3) The OPEN INPUT and OPEN OUTPUT statements are not used to create the C2 variant.

Letter	PCU
A(21)	01
B(22)	02
C(23)	03
D(24)	40
E(25)	41
F(26)	42
G(27)	43
H(30)	00

(4) COBOL H also provides the option of designating these addresses by a two-digit octal number (integer-3); e.g., 41. Thus, CARD-READER E can also be designated as CARD-READER 41.

9. When more than one reel may exist in a file, the MULTIPLE REEL entry can be used for documentary purposes.

10. If the RESERVE entry is not made, double buffering is provided for the SELECTed file. The use of

RESERVE NO ALTERNATE AREA(S)

results in a single buffering for the SELECTed file.

I-O-CONTROL

B. I-O-CONTROL

FUNCTION: To specify the input/output techniques, the points at which reruns are to be established, and the memory area which is to be shared by different files.

FORMAT:

option 1:

```
[I-O-CONTROL.  COPY library-call._ ]
```

option 2:

```
[ I-O-CONTROL. [ APPLY { NO-TAPE-MARK     } ON file-name-1 [, file-name-2 ...]]
                       { H-200-SPECIAL    }
                       { TRAILING-COUNT   }
                       { SHORT-GAP        }
  [; SAME AREA FOR file-name-5, file-name-6 [, file-name-7 ...]]
  [; MULTIPLE FILE TAPE CONTAINS file-name-8, file-name-9 [, file-name-10 ...]]
  [; RERUN [ ON file-name-11] EVERY { END OF REEL      } OF file-name-12] . ]
                                    { integer-1 RECORDS }
```

TECHNICAL NOTES:

1. Option 1 is used only when the COBOL library contains the entire I-O-CONTROL paragraph (except the paragraph-name) as shown in option 2.

2. The

   APPLY H-200-SPECIAL ON...

   clause is used only for card-read and card-punch files. When this clause is specified, cards in the file named are read/punched in special mode at object time.

3. The

   APPLY TRAILING COUNT ON ...

   clause causes the generation of object coding to create magnetic tape files that can be processed by Series 200 programs that utilize the read backward instruction.

4. The

   APPLY SHORT-GAP ON ...

   clause may be applied to any output file (tape, punch or printer), but is effected only when the file is written on tape at object time. The data is written on tape with short inter-record gaps rather than the standard .75-inch gap.

5. The SAME AREA... clause is used to specify that two or more files are to use the same area of memory for processing. In object program execution, however, no more than one such file can be OPEN at the same time.

6. The MULTIPLE FILE clause must be used when more than one file shares the same physical reel of tape. Regardless of the number of files on a single tape, only those files which are used in the object program need be specified. All file-names must be listed in consecutive order. Note that files extending over more than one reel (multiple-reel files) cannot be specified by a MULTIPLE FILE clause.

7. No more than four **MULTIPLE FILE** clauses can be given in one source program.

8. No more than one RERUN clause may appear within a source program. The RERUN clause may not be specified if NO SEGMENTATION is used in the OBJECT-COMPUTER paragraph. Rerun dumps must not be taken on BCD files. See Section VIII for RERUN operating instructions.

9. File-name-11 must be an output file ASSIGNed to tape with STANDARD labels, and the file must be OPENed before the RERUN point is reached. If file-name-11 is not specified, file-name-12 must conform to these rules. If file-name-11 has been used, file-name-12 may be a file on tape, card reader, card punch, or printer.

10. If

    ON file-name-12

    is specified and

    END OF REEL

    is also specified, file-name-12 must be a tape file. The rerun information is then placed after the file identification block on the next reel of the file.

11. If

    NO-TAPE-MARK

    is specified, the tape mark following the header tape label (BCD files) is suppressed.

SECTION V
THE DATA DIVISION

General

Data to be processed falls into three 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.
3. Constant data which is defined by the programmer.

Consequently, the DATA DIVISION consists of three 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.
3. CONSTANT SECTION, in which the fixed, the constant, values desired by the programmer are stated.

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

The FILE SECTION [illegible] the WORKING-STORAGE SECTION and the [illegible]

DATA DIVISION ENTRIES

File Description Entry

A. General

A File Description entry contains information pertaining to the physical aspects of a file. In general, it may include the following: the manner in which the data is recorded on the file, the size of the logical and physical records, the names of and values of the label records contained in the file, the names of the data records which comprise the file, and finally the keys on which the data records have been sequenced. 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.

B. File Description Entry Formats

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 their recommended order of appearance.

FILE DESCRIPTION

B. 1. FILE DESCRIPTION

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

SECTION FORMAT:

(FILE DESCRIPTION)

Option 1:

FD file-name COPY library-name .

Option 2:

FD file-name
[; RECORDING MODE IS { H-200 / BCD }]
Binary Coded Decimal.

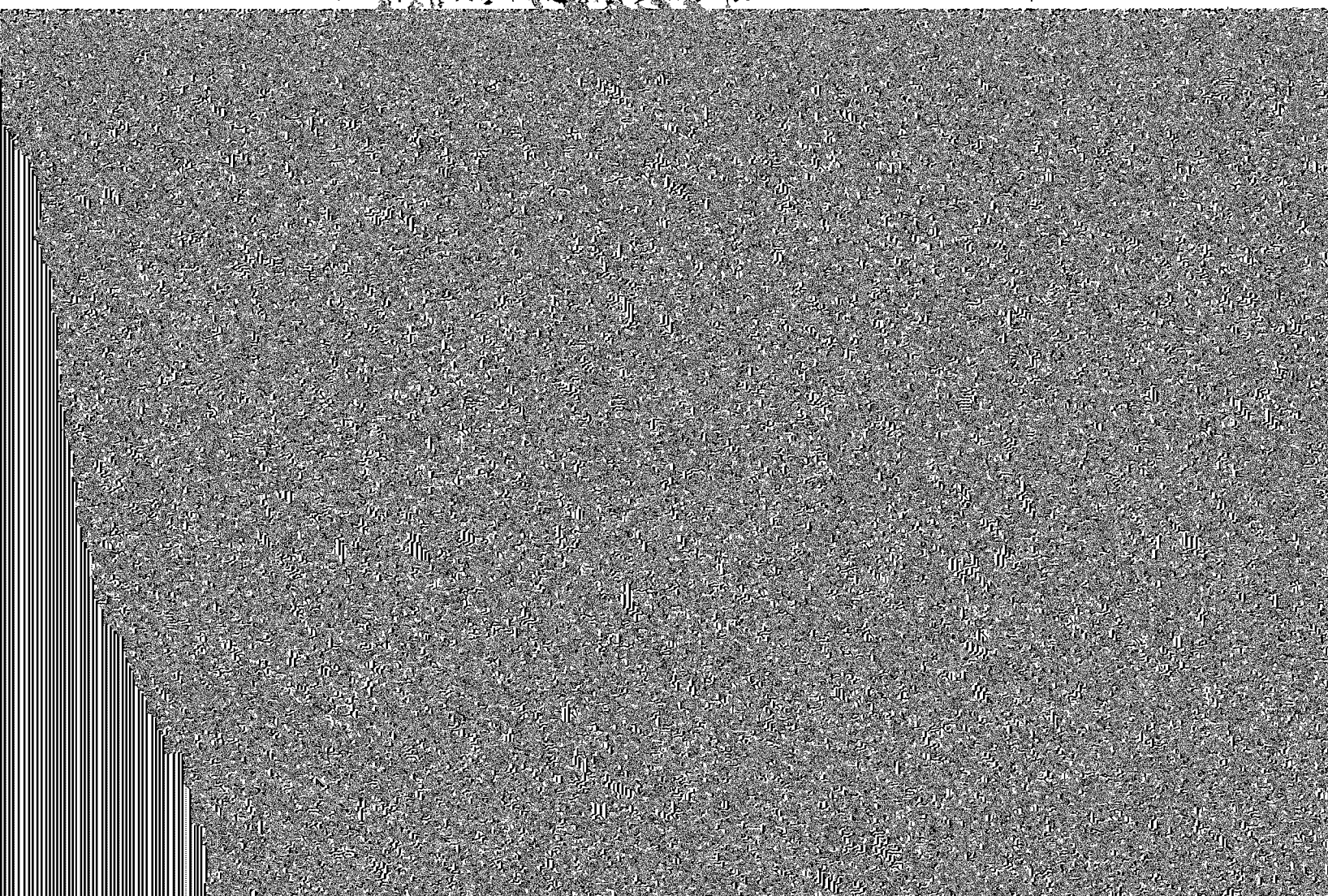
[; FILE CONTAINS ABOUT integer-1 RECORDS]

[; BLOCK CONTAINS [integer-2 TO] integer-3 { RECORD(S) / CHARACTERS }]

[; RECORD CONTAINS [integer-4 TO] integer-5 CHARACTERS]

; LABEL RECORDS ARE { STANDARD / STANDARD-80 / STANDARD-120 / OMITTED }
Label Length 80 bytes (Honeywell Standard)
Label Length 120 bytes (IBM ")

[NAME OF IDENTIFICATION IS { literal-1 }]
1-10 chars.



BLOCK SIZE

B. 2. BLOCK SIZE

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

FORMAT:

$$\left[; \underline{\text{BLOCK}} \text{ CONTAINS } \left[\text{integer-2 } \underline{\text{TO}} \right] \text{integer-3} \left\{ \begin{matrix} \underline{\text{RECORD(S)}} \\ \underline{\text{CHARACTERS}} \end{matrix} \right\} \right]$$

TECHNICAL NOTES:

1. Integer-2 and integer-3 must each be a numeric literal with an integral value.
2. This clause is required when the block is to contain more than one logical record.
3. The size of a block defined by this clause cannot exceed the memory size of the object computer. Note that no record can exceed 4,095 characters in length.
4. When this clause is used, the size of the block must be stated as the number of records per block except when the file records are of variable length. With file records of variable length, block size may be stated in terms of records or characters.
   a. If the records vary in length and the block size is given in terms of characters, the object code generated is such that the greatest possible number of integral records is placed in each block. Blocks are not padded but vary in size, depending on the size of the records placed in them by successive WRITEs. The last block of the file contains no padding. (This note also applies to BCD tapes.)
   b. If the records vary in length and the block size is given in terms of records, the object code generated is such that exactly integer-3 records appear in every block of the file. Since the records vary in length, the blocks also vary in length. The last block of the file (or of an intermediate reel of the file) contains as many padding records of maximum size as are needed to supply integer-3 records for the block. Padding records are always equal in size to the largest file record and consist of all octal 77 characters. (This note does not apply to BCD tapes.)
5. Integer-2 is for documentary purposes only and is not used by the compiler.
6. When the CHARACTERS option is used, the block size is specified in terms of the number of six-bit characters contained within the block.
7. If a peripheral file (i. e. , a file assigned to the card reader, to the card-punch, or to the printer) is being described, this clause should be in the form:

   BLOCK CONTAINS 1 RECORDS

Note for use of variable length records.

COPY

B. 3. COPY

FUNCTION: To obtain a file description entry from the COBOL library.

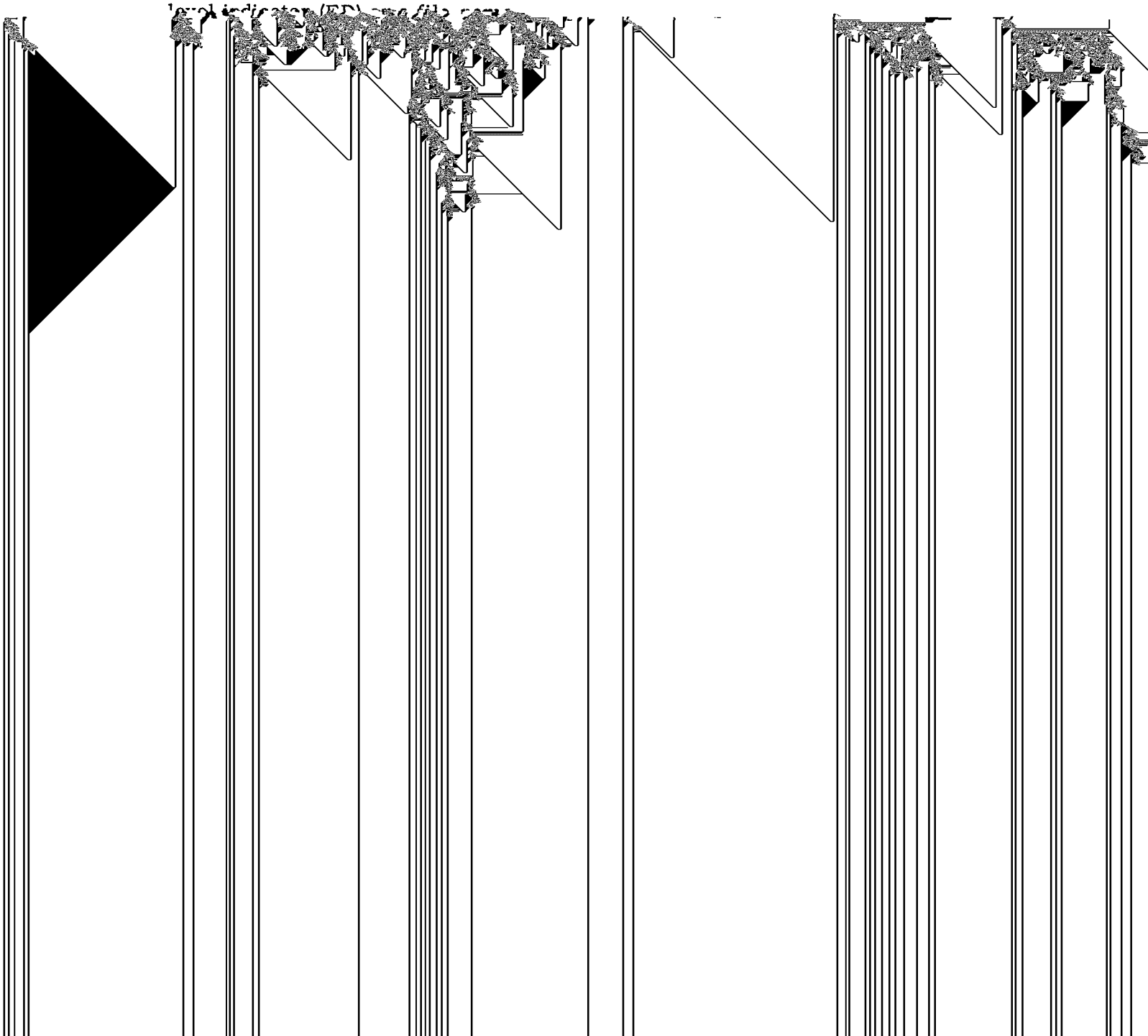
FORMAT:

(FD file-name) COPY library-name.

TECHNICAL NOTES:

1. COPY is used when the COBOL library contains the entire File Description entry.

2. During compilation, the COPY clause is replaced by the sequence of clauses within the library-name entry. Thus, the level indicator (FD) and the file-name which precede the COPY clause are retained and the library entry is identically COPYed; therefore, the library entry must not contain either a



B. 4. DATA RECORD(S)

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

FORMAT:

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

TECHNICAL NOTES:

1. The presence of more than one record-name indicates that the file contains more than one type of data-record. The order in which they are listed is not significant.
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. Subscripting of these data-names is not permitted.
4. This clause is used for documentation purposes only; record-names are never checked against the 01 level record-names.

FD

B. 5. FD

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

FORMAT:

FD 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 follow the FD entry.
3. The file-name must be SELECTed in a FILE-CONTROL entry.

B.6. FILE

FUNCTION: To indicate the approximate number of logical records in a file.

FORMAT:

[; FILE CONTAINS ABOUT integer-1 RECORDS]

TECHNICAL NOTES:

1. This clause is for documentary purposes only.
2. Integer-1 must be a numeric literal with an integral value.

B. 7. LABEL RECORDS

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

FORMAT:

$$; \underline{\text{LABEL RECORD(S)}} \quad \left\{ \begin{matrix} \text{ARE} \\ \text{IS} \end{matrix} \right\} \quad \left[\begin{matrix} \underline{\text{STANDARD}} \\ \underline{\text{STANDARD-80}} \\ \underline{\text{STANDARD-120}} \\ \underline{\text{OMITTED}} \end{matrix} \right]$$

TECHNICAL NOTES:

1. This clause is required in every File Description entry.
2. When OMITTED is specified, the file is understood to conform to none of:
   a. The label block conventions for Series 200 magnetic tape files, except that the tape containing (or to contain) the file may have a beginning-of-tape label; or
   b. The label conventions associated with punched card files.
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. It may be followed by the

   DATE-WRITTEN IS...

   clause.
5. STANDARD-80 and STANDARD are considered to be identical.
6. STANDARD-120 indicates 120-character labels, and should only be specified for files with RECORDING MODE IS BCD.
7. When OMITTED is specified and the file has RECORDING MODE IS BCD, no label should appear before the first block of data on tape.

RECORD size

B. 8. RECORD size

FUNCTION: To specify the size of data records.

FORMAT:

[; RECORD CONTAINS [integer-4 TO] integer-5 CHARACTERS]

TECHNICAL NOTES:

1. Integer-4 and integer-5 must each be a numeric literal with an integral value.
2. The size of each data record is completely defined within the Record Description entries; therefore, this clause is never required. When present, however, the following notes apply:
   a. Integer-5 may not be used by itself unless all the data records in the file have the same size. In this case integer-5 represents the exact number of characters in the data record. If integer-4 and integer-5 are both shown, they refer to the minimum number of characters in the smallest size of data record and the maximum number of characters in the largest size data record, respectively.
   b. The size is specified in terms of the number of standard characters contained within the logical record, regardless of the types of characters used to represent the items within the logical record.
3. The maximum size of elementary or group items is 4095 characters. See "Subscripting" (Section I) for a method of circumventing this limitation.
4. This clause is for documentary purposes only.

B.9 RECORDING MODE

FUNCTION: To specify the format or organization of data on external media.

VERB FORMAT:

$$\left[;\ \underline{\text{RECORDING}}\ \text{MODE IS}\ \left\{\begin{matrix}\underline{\text{H-200}}\\ \underline{\text{BCD}}\end{matrix}\right\}\right]$$

TECHNICAL NOTES:

1. The H-200 option is for documentary purposes only.
2. The BCD option implies even-parity tape. (See BLOCK SIZE clause, LABEL RECORDS clause, and Section VII.)

B. 10. SEQUENCED ON

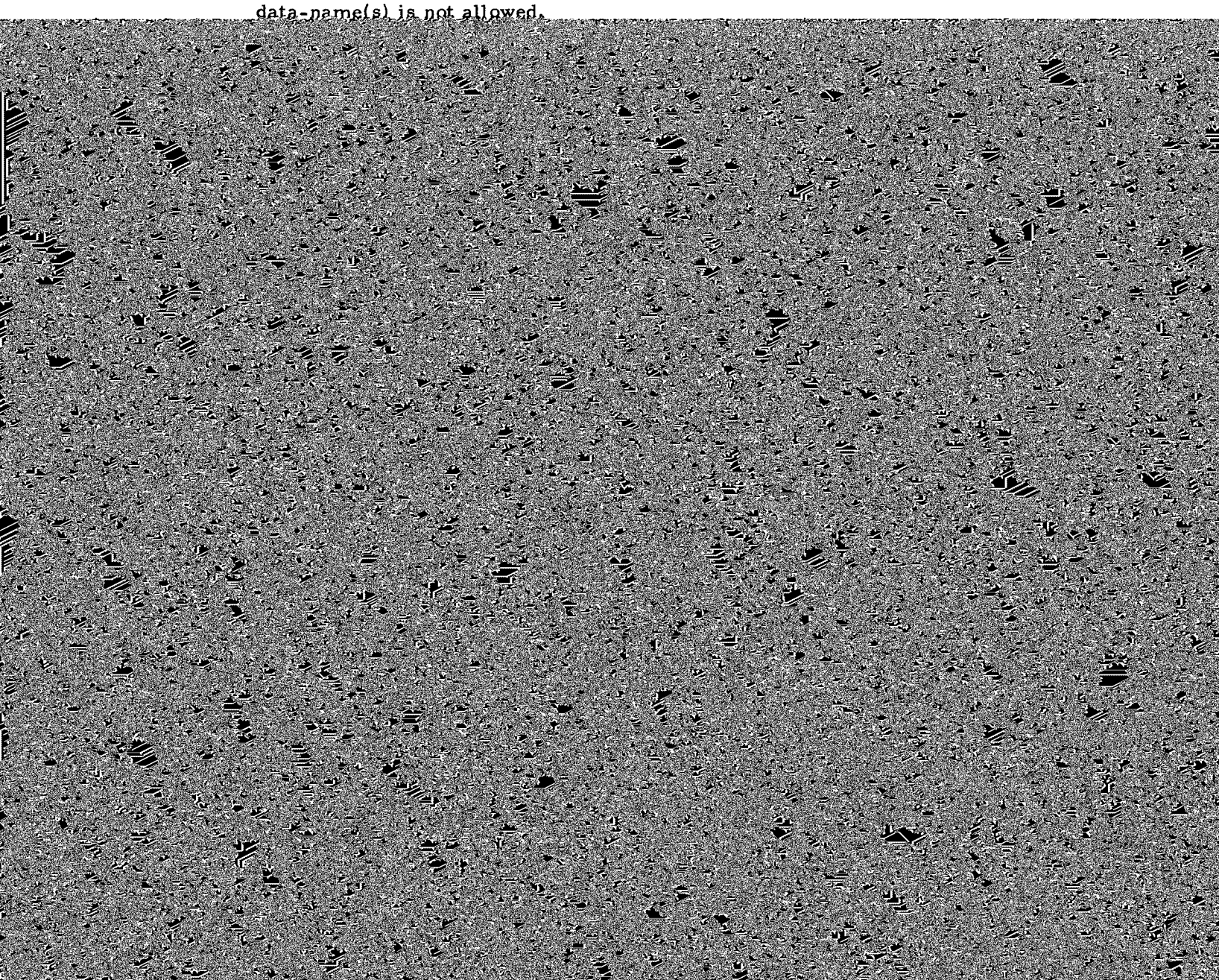
FUNCTION: To indicate the keys on which data records are sequenced.

FORMAT:

[; SEQUENCED ON data-name-3 [, data-name-4]]

TECHNICAL NOTES:

1. This clause is used only for documentation; however, when used, it must be used as shown.
2. Data-name-3 represents the major sorting key, data-name-4 the lowest sorting key. No more than two fields can be named in a SEQUENCED ON clause.
3. The total number of contiguous or non-contiguous computer character positions containing the fields named in a sequenced on clause cannot exceed two.
4. Data-name-3 (data-name-4) must be common to all data records in the file and must occupy the same position(s) in each record. Subscripting of the data-name(s) is not allowed.



B. 11. VALUE OF

FUNCTION: To particularize the description of an item in the label records associated with a magnetic tape or a punched card file.

FORMAT:

$$\left[\underline{\text{VALUE OF IDENTIFICATION}}\ \text{IS}\ \left\{\begin{matrix}\text{data-name-1}\\ \text{literal-1}\end{matrix}\right\}\ \left[\ ;\ \underline{\text{DATE-WRITTEN}}\ \text{IS}\left\{\begin{matrix}\text{data-name-2}\\ \text{literal-2}\end{matrix}\right\}\right]\right]$$

TECHNICAL NOTES:

1. This clause must be given when the

   LABEL RECORDS ARE STANDARD

   clause is given and must be in the minimum form

   VALUE OF IDENTIFICATION IS literal-1

   which will cause 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 checked for in that field of the label records of an input file at object time.

2. The DATE-WRITTEN object can be specified for input and output files. When specified for input files, the value is checked against the input Beginning of File label.

3. The contents of data-name-1, or the value stated in the non-numeric literal-1 must be from 1 through 10 alphanumeric characters (space characters are allowed). If less than ten characters are used, spaces are placed to the right of the last character to make a ten-character field. The size of data-name-1 must be 10.

4. Data-name-2 content (or the value of numeric literal-2) must be five-numeric characters. Data-name-2 cannot be more than five COBOL characters in length.

5. Data-name-1 cannot be more than ten COBOL characters in length.

6. Data-name-1 and data-name-2 cannot be qualified.

Record Description Entry

A. General

A detailed data description consists of a set of entries. Each entry 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 may be required.

1. A level number which shows the relationship between this and other units of data.
2. A data-name.
3. The SIZE in terms of the number of standard data format (six bit) characters.
4. The dominant USAGE of the data.
5. The number of consecutive occurrences (OCCURS) of elements in a table or list.
6. The RANGE of values which the data may assume.
7. The CLASS or type of data; i.e., alphabetic, numeric or alphanumeric.
8. The location and type of SIGN.
9. Location of an actual or an assumed radix point.
10. Location of editing symbols such as dollar signs and commas.
11. Justification and synchronization of the data. (JUSTIFIED, SYNCHRONIZED).
12. Special editing requirements such as zero suppression and check protection.
13. Initial VALUE of a working-storage item or the fixed VALUE of a constant.

An entry which defines a unit of data must not be contradicted by a subordinate entry. Thus, once the CLASS is defined, it applies to all subordinate entries and need not be respecified in the subordinate entries. However, when CLASS is defined as ALPHANUMERIC, subordinate entries can nay

RECORD DESCRIPTION

B. 1. RECORD DESCRIPTION

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

SECTION FORMAT:

Option 1: level-number data-name [; REDEFINES...] ; COPY... .

Option 2: level-number { data-name-1 / FILLER }

[; REDEFINES data-name-2]

[; SIZE IS { integer-1a { CHARACTERS / DIGITS } / integer-1 TO integer-1a { CHARACTERS / DIGITS } [DEPENDING ON data-name-3] }]

[; USAGE IS { DISPLAY / DISPLAY-1 / DISPLAY-2 / COMPUTATIONAL / COMPUTATIONAL-1 / COMP / COMP-1 }]

[; OCCURS [integer-2 TO] integer-3 TIMES [DEPENDING ON data-name-4]]

[; SIGNED]

[; SYNCHRONIZED { LEFT / RIGHT }]

[; CLASS IS { NUMERIC / ALPHABETIC / ALPHANUMERIC / AN = alphanumeric. }]

[; (Decimal-) POINT LOCATION IS { LEFT / RIGHT } integer-4 PLACES]

[; PICTURE IS { any allowable combination of PICTURE characters }]

[; JUSTIFIED { LEFT / RIGHT }]

[; RANGE IS literal-1 THRU literal-2]

[; { ZERO SUPPRESS / CHECK PROTECT / FLOAT DOLLAR SIGN } [LEAVING integer-5 PLACES]] [BLANK WHEN ZERO]

[; VALUE IS literal-3] .

Option 3: 88 condition-name $\left\{\begin{matrix}\underline{\text{VALUE}}\text{ IS} \\ \underline{\text{VALUES}}\text{ ARE}\end{matrix}\right\}$ literal-1 $\left[\underline{\text{THRU}}\text{ literal-2}\right]$ $\left[\text{literal-3 } \left[\underline{\text{THRU}}\text{ literal-4}\right]\right]$.

TECHNICAL NOTES:

1. For an explanation of the reference format used in the DATA DIVISION, see page II-4.
2. Those clauses which begin with SIGN, SYNCHRONIZED, POINT, PICTURE, JUSTIFIED, and RANGE, as well as the editing clauses, must not be specified except at the elementary item level.
3. The clauses can be written in any order with the exception of REDEFINES. REDEFINES, when used, must immediately follow data-name-1.
4. All semicolons are optional in the Record Description entry.
5. Detailed discussions of each of the clauses is given in the succeeding pages.
6. The COPY option is a Language Preprocessor element. See Section XI for a complete description.
7. The editing options are Language Preprocessor elements. See Section XI for a complete description.

B. 2. CHECK PROTECT

FUNCTION: To permit suppression of non-significant zeros and commas, replacing each zero and each comma suppressed with an asterisk (*).

For format and technical notes, see "Editing Clauses" (Section XI).

CLASS

B.3. CLASS

FUNCTION: To indicate the type of data being described.

FORMAT:

$$(\text{data-name-1}\ldots)\left[\ ;\ \text{CLASS IS}\ \left\{\begin{array}{l}\underline{\text{NUMERIC}}\\ \underline{\text{ALPHABETIC}}\\ \underline{\text{ALPHANUMERIC}}\\ \underline{\text{AN}}\end{array}\right\}\ \right]$$

TECHNICAL NOTES:

1. AN is an abbreviation for ALPHANUMERIC.

2. The CLASS clause should be written only at the elementary level. If the CLASS clause is written at a group level, it is useful only as documentation. ALPHABETIC or NUMERIC items within an ALPHANUMERIC group are not considered contradictory.

3. A data item can be called NUMERIC only if it is composed solely of characters chosen from the numerals 0 through 9, an operational plus or minus sign, and an implied decimal point. If there is no sign associated with the data, the data is considered positive. If the data is NUMERIC and no assumed decimal point is indicated, the data is considered to be an integer.

   A distinction must be made between the terms NUMERIC and numeric as applied to data items. An item is said to be NUMERIC when it can be determined at compilation time that its CLASS is NUMERIC by its own Data Description (e.g., CLASS, PICTURE, SIGNED, and USAGE) or by the Data Descriptions of items in its hierarchy. An item is said to be numeric when it can be determined at object program time that its contents are, in fact, numeric.

4. ALPHABETIC describes data which contains any combination of the twenty-six (26) letters of the English alphabet and the space. No other characters and no numerals can be used.

5. ALPHANUMERIC describes data which may contain any allowable character in the Honeywell character set, including alphabetics and numerics. Thus, data which is ALPHABETIC or NUMERIC is also ALPHANUMERIC.

6. If a PICTURE is given, the CLASS clause is unnecessary. If both are used, the CLASS clause is ignored.

7. If the CLASS of an elementary item cannot be determined from any clause in the item's Record Description or from the Record Description of any group to which the item belongs, the CLASS of the item is assumed to be ALPHANUMERIC.

COPY

B. 4. COPY

FUNCTION: To duplicate within this record all or part of another Record Description contained in a library.

For format and technical notes, see "Language Preprocessor" (Section XI).

data-name
FILLER

B.5. data-name/FILLER

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

FORMAT:

(level-number) { data-name / FILLER }

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. | i.e. NOT .05 FILLER.

2. The key word FILLER is used to name an item in a record. Under no circumstances can a FILLER item be referenced directly. The data associated with each FILLER entry must be left unaltered by the object program.

B.6. Editing Clauses

FUNCTION: To permit suppression of non-significant zeros and commas; to permit floating dollar signs or check protection; and to permit the blanking of an item when its value is zero.

FORMAT:

To leave a set number of figures untouched to the left of the decimal point.

```
[ ; { ZERO SUPPRESS      } [ LEAVING integer-4 PLACES ] ] BLANK WHEN ZERO
    { CHECK PROTECT      }
    { FLOAT DOLLAR SIGN  }
```

TECHNICAL NOTES:

1. When CHECK PROTECT is specified, leading zeros and commas are replaced by asterisks.
2. CHECK PROTECT overrides BLANK WHEN ZERO. When the BLANK WHEN ZERO option is used, the item contains nothing but spaces if the value of the item is zero, thus overriding ZERO SUPPRESS and FLOAT DOLLAR SIGN.
3. For remaining technical notes, see "Language Preprocessor" (Section XI).

B.7. FLOAT DOLLAR SIGN

FUNCTION: To permit suppression of non-significant zeros and commas, the last zero or comma suppressed having a dollar sign ($) put in its place.

For format and technical notes, see "Editing Clauses" (Section XI).

B.8. JUSTIFIED

FUNCTION: To specify non-standard positioning of a data item when less than the maximum number of characters may be present.

FORMAT:

(data-name-1...) [; JUSTIFIED RIGHT]

TECHNICAL NOTES:

1. The standard rules of data positioning within a numeric or numeric-edited receiving field are:
   a. Numeric data is aligned by decimal point with zero fill on either end as required.
   b. When no assumed decimal point is specified, numeric data is right-justified with zero fill.
2. The standard rules for positioning within a non-numeric receiving field are: alphabetic and alphanumeric data is left-justified with space fill.
3. The JUSTIFIED clause is required only when the standard rules of positioning in a non-numeric receiving field are not followed. When non-standard positioning of numeric data is desired, a non-numeric receiving field may be used.
4. When the

   JUSTIFIED RIGHT

   clause is specified, data being received by this field will be justified right with space fill from the left.

level-number

B.9. level-number

FUNCTION: To show the hierarchy of data within a logical record. To identify entries for condition-names, non-contiguous constants and working storage items.

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 49, 77, and 88.

3. Level numbers must not be used to construct a hierarchy in which any data-name could be qualified at more than ten levels.

4. The level-number 01 indicates the first entry in each Record Description. READ File name This corresponds to the logical record on which the ~~READ~~ and WRITE verbs WRITE Record name 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 (Field. i.e. not broken down.)
```

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

```
02 employee-name (Group header. i.e. broken down.)
   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
            06 1st-character
            06 last-character
         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,

B.10. OCCURS

OCCURS

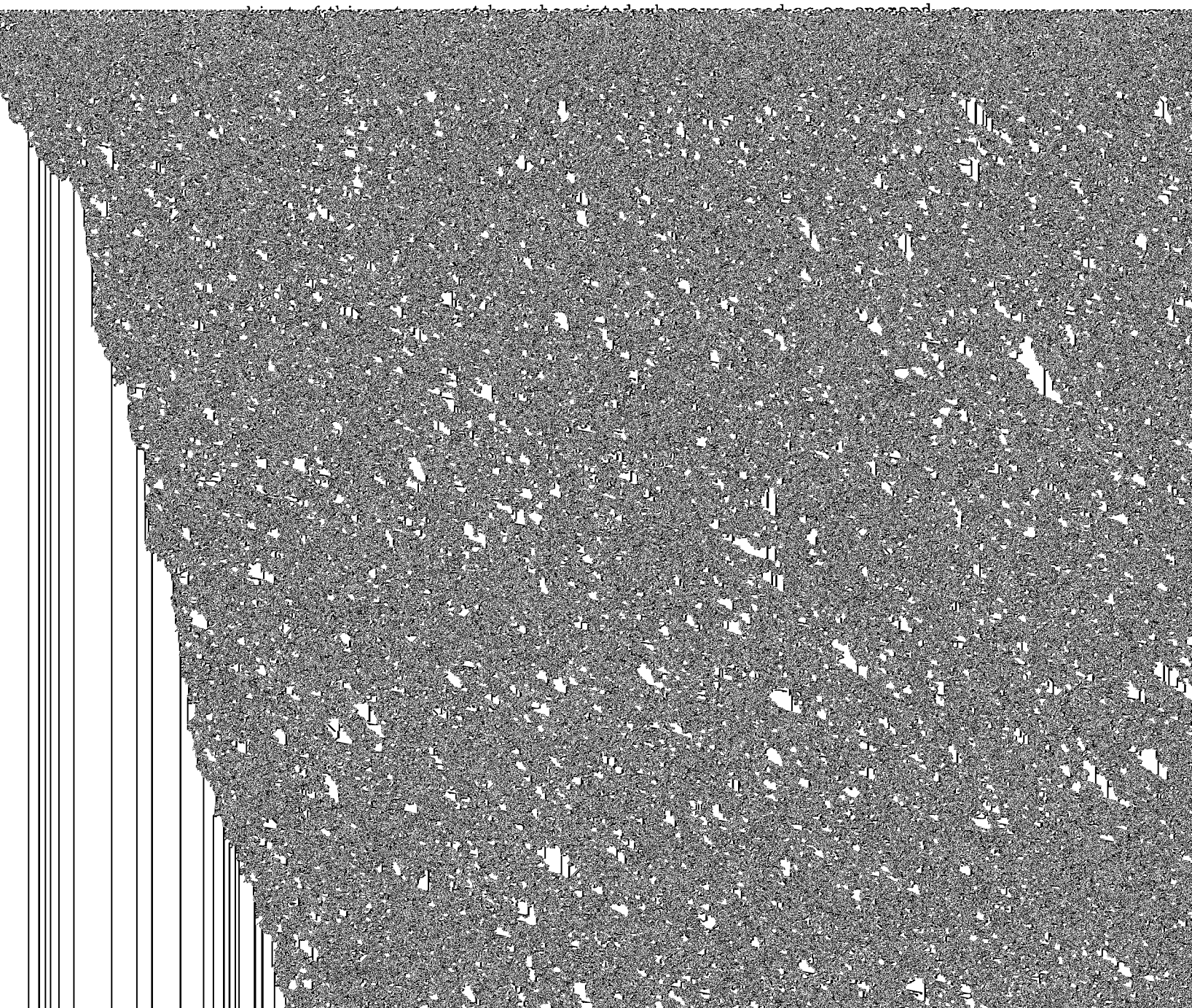
FUNCTION: To eliminate the need for separate entries for repeated data and to supply information required in the application of subscripts.

FORMAT:

(data-name-1...) [; OCCURS [integer-2 TO] integer-3 TIMES [DEPENDING ON data-name-4]]

TECHNICAL NOTES:

1. Integer-2 and integer-3 must be numeric literals with positive integral values.
2. This clause cannot be specified in a Record Description entry that:
   a. has an 01 level number,
   b. describes a variable-length item (i.e., the OCCURing item must be of fixed length).
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



c. When the record is part of an output PRINT file, it cannot OCCUR more times than would require 132 character positions.

8. 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

PICTURE

B.11. PICTURE

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

FORMAT:

(data-name-1...) [; PICTURE IS character-string]

TECHNICAL NOTES:

1. A PICTURE clause can only be used to describe an elementary item.
2. The editing characters or symbols that can be used in a PICTURE clause are:
   a. Data Characters

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

      Operational Symbols

      S indicates the presence of a sign
      V assumed decimal-point location
      P assumed decimal-point scaling position

      Zero Suppression Characters

      Z standard zero-suppression (replacement of leading zeros by blanks or spaces)
      * check protection (replacement of leading zeros by asterisks)

      Insertion Characters

      $ dollar sign (floating and replacing last leading zero when more than one appears)
      , comma
      . actual decimal point
      B blank or space
      0 zero

      Report Signs

      + positive sign } (floating and replacing last leading zero when
      - negative sign } more than one appears)
      CR credit
      DB debit

      Picture Abbreviation Characters

      (
)
      n (where n is a positive integer < 4096).

3. There can be no more than 30 of the above characters in a PICTURE (see notes 7, 12, and 13 for restrictions because of the size of the item being described). Thus, AAAAAAAAAA contains 10 PICTURE characters, A(10) contains 5 PICTURE characters, 9(30) P(10) contains 10 PICTURE characters.
4. All characters other than the operational symbols (S V and P) are counted in the size of the item described in the PICTURE clause.

5. The number of characters in a PICTURE must not be confused with the size of the item being described. For example 9(30)P(20)V contains 11 characters but describes an item containing 30 characters.

6. In the following notes, a "numeric" item is one whose PICTURE consists of S V (P) 9 and either high or low order 0's. Specifically, a PICTURE containing one or more 0's is considered "numeric" only if all the 0's precede or all the 0's follow any 9's in the PICTURE; otherwise, the item is considered to be "numeric edited."

7. 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 data item. (See MOVE verb.)

   a. NUMERIC characters
   N - numeric (see Move)

      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 written as 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 the data item. The PICTURE of a data item cannot contain more than one assumed decimal point. Since an extreme right V in a PICTURE is considered redundant, when V is not specified for a numeric data item, the decimal point is assumed to the right of the PICTURE description, so long as the PICTURE does not contain leftmost P's. If it does contain leftmost P's the decimal point is assumed to their left. No more than thirty-one 9's can appear to the right of a V.

      P Indicates an assumed decimal scaling position and is used to specify the location of an assumed decimal point when the point is not within the number that appears in the data item. The scaling position character P is not counted in the size of the data item. Scaling position characters are not counted in determining the maximum number of digit positions (29 for numeric edited items; 63 for numeric items with non-zero point locations, unlimited otherwise) which appear as operands in arithmetic statements. The scaling position character P can appear only to the left or to the right as a continuous string of P's within a PICTURE description; since the scaling position character P implies an assumed decimal point (to the left of the P's if P's are the leftmost PICTURE characters or to the right of the P's if P's are the rightmost PICTURE characters), the assumed decimal point symbol V is considered redundant as either the leftmost or the rightmost character within such a PICTURE description. Also, in a receiving field, all P's must appear to the right or to the left of any receiving character positions in the PICTURE. Only insertion characters can precede leftmost or follow rightmost P's. The maximum num-

ber of rightmost P's that can appear is thirty-one. The maximum number of leftmost P's and 9's that can appear together is thirty-two.

0 Indicates the insertion character zero 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 0, the numeric data is aligned by decimal point within the receiving character positions independently of the insertion characters. The insertion characters are placed in the receiving data item regardless of the nature of the sending data item. If a numeric data item with a PICTURE containing one or more 0's is moved to another data item, each data item character position is considered in the sending data item as part of the value for decimal point alignment purposes. There is a maximum of 15 zeros allowed to precede or a maximum of 15 zeros allowed to follow, the explicit or implicit decimal point of an item.

b. ALPHABETIC characters
AB - alphabetic (see MOVE)

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 left justified within the receiving character positions independently of the insertion characters. 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.

c. ALPHANUMERIC characters
AN - alphanumeric (see MOVE)

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 considered and treated as if the entire PICTURE consisted of all X's.

AE - alphanumeric edited (see MOVE)

A PICTURE description containing at least one of the insertion characters B or 0 and at least one of the character X, or a PICTURE description containing at least one of the characters X or A, is considered an alphanumeric edited data item.

NE - numeric edited (see MOVE)

Numeric edited implies:
1) The data item being described is alphanumeric.
2) The data item being described is a numeric edited data item.
3) The data item can only receive data which is numeric in content.

9, V, and P have been defined under numeric characters above. 0 and B are repeated in definition here to indicate special editing cause and effect.

B, 0 — Indicate the insertion characters comma, space (blank) and zero respectively; 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 or replacement of zeros by spaces (Z) and check protection (*) indicate that suppression of leading insertion characters also takes place with associated space or asterisk replacement.

$$, ++, -- — The floating characters, floating dollar ($$), floating plus (++), and floating minus (--), "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. A PICTURE description must not terminate with the symbol "," unless immediately followed by one of the punctuation characters, comma, semicolon, or period.

. — 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. A data item cannot contain more than one assumed or actual decimal point, i.e., the symbols "V" and "." are mutually exclusive within a PICTURE description. A PICTURE description must not terminate with the symbol ".", unless immediately followed by one of the punctuation characters, comma, semicolon, or period.

eg. Picture is $99.; value is 02. (i.e. $99. wanted rather than $99 only)

+, -, CR, DB — Indicate the editing sign control characters. As fixed insertion characters, the "+" and "-" symbols can be used only at the beginning or at the end of a PICTURE; "CR" and "DB" can be used only at the extreme right end of a PICTURE and represent two character positions when counted in the size of the data item. When the fixed insertion characters "+" or "-" are used at the beginning, they must be the leftmost character in the PICTURE. As floating characters, the "+" and "-" characters are written from the extreme left to represent each leading numeric character position into which the editing sign may be floated. A single editing sign will be placed in the least significant position shown by the character "+" or "-" in the PICTURE description (including the insertion characters ",", "B", or "0") immediately preceding the first non-zero digit in the data, or the decimal point "V" or ".", whichever is encountered first. One exception is that if all data character positions in the PICTURE contain the floating characters "+" or "-", then the entire data item will consist of spaces when the value of the data item is zero. A floating "+" or "-" may appear to the right of a decimal point in a PICTURE only if all character positions are represented by "+'s" or "-'s". The

data item must contain at least one more editing sign character position than the maximum number of significant digits in the associated source data item. Editing signs are counted in the size of the data item. Depending on the value of the data item associated with the PICTURE, the display character signs outlined in the following table are produced. "+", "-", "CR", "DB"

will be placed in the least significant position shown by the character "$" in the PICTURE (including the insertion characters ", ", "B", and "0") immediately preceding the first non-zero digit in the data, or the decimal point "V" or ",", whichever is encountered first. One exception is that if all the numeric character positions in the PICTURE contain the floating character "$", then the entire data item will consist of spaces when the value of the data item is zero. The data item must contain at least one more "$" character position than the maximum number of significant digits in the associated source data item. A "$" may appear to the right of a decimal point only if all numeric character positions are represented by "$'s". The dollar sign is counted in the size of a data item.

Otherwise the $ sign will be overlaid.

8. A PICTURE must consist of 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)

9. Only one type of floating replacement character, i.e., "$", "+", "-", "*", or "Z" can be used within a given PICTURE description. The replacement characters "*" or "Z" may be preceded by a fixed "$"; "$" (fixed or floating), "Z" or "*" may be used with a fixed "+" or "-". A PICTURE character "9" can never appear to the left of a floating or a replacement character.

10. An integer which is enclosed in parentheses following the symbols

    A X 9 P Z * B $ 0 + -

    indicates the number of consecutive occurrences of the symbol. (E.g., P(10)9(2) and P(10)99 and PPPPPPPPPP99 are all equivalent.) Note that the following symbols may appear only once in a given PICTURE:

    S V . CR DB

11. If editing clauses are used in conjunction with a PICTURE clause, the two sets of clauses must not be contradictory. The PICTURE character "*" is mutually exclusive with the editing clause BLANK WHEN ZERO.

12. If any of the following characters occurs in a PICTURE, the size of the item being described cannot exceed 29 characters:

    Z * B 0 + - $, . CR DB

13. Within the limits of note 12., above, the highest parenthesized number that can follow

    A X 9

    is ~~4096~~ 4095.

    The highest parenthesized number that can follow

    P

    is 32.

    The highest parenthesized number that can follow

    Z * + - $

    is 29.

The highest parenthesized number that can follow

B

is 28.

14. All legal COBOL PICTUREs are handled correctly, but certain inefficiencies arise from peculiarities in the execution of the MCE instruction. The Series 200 COBOL programmer should avoid, whenever possible, writing PICTUREs which vary from the following rules:

a. It must not contain a "DB." or "+."

b. It must not contain more than one "-."

c. The rightmost suppression "$" or "*," if present, must be immediately preceded by the same character.

d. The rightmost "Z," if present, must not be immediately preceded by a "$."

e. The rightmost suppression character "$," "*," or "Z," if present, must not be followed by a "," or "B."

f. It must not specify BLANK WHEN ZERO.

g. It must not imply non-simple zero-editing. Zero-editing may be divided into two categories, simple and non-simple. A field is defined to be simple-edited if the editing characters present are all high-order zeroes. In such a case, the receiving field will simply be filled at the appropriate end with the correct number of zeroes. In all other cases of zero-editing, a special code must be placed into the MCE constant where each zero is to be inserted. This is necessary because zero has a special meaning to the MCE instruction. After editing, the special code characters must be replaced by zeroes.

15. For source language PICTUREs that contain a floating "+," "-," or "$," and at least one insertion blank, the following departure from COBOL rules will occur at object time: the floating character will float up to, but not into the leftmost blank position and no further. Therefore, any floating "-," "+," or "$," to the right of an insertion blank is ineffective and will be treated as a 9. Also, an "*" will float leftward into an insertion blank, or into a blank created when a field with a plus sign is moved into a field having a PICTURE with a high-order insertion "-."

16. The following table summarizes the rules for PICTURE symbol use.

PICTURE CHARACTER	MEANING	RESTRICTIONS	REQUIRED PICTURE POSITION
9	A NUMERIC character.	none	none
S	An operational sign	a. Can occur only once in any PICTURE b. Can be used in PICTURES containing only V P 9 and/or 0	Must be the leftmost character in the PICTURE.
V	An assumed decimal point.	a. Only one V can occur in any PICTURE. b. None of the characters . A X can occur in the same PICTURE.	If used with P's, V must precede leftmost P's or must follow rightmost P's; i.e., VPPP9999 or 9999PPPV

PICTURE CHARACTER	MEANING	RESTRICTIONS	REQUIRED PICTURE POSITION
*	Replacement of leading zeros and commas by asterisks (the equivalent of check protect); if both to the right and left of an actual decimal point (.), the decimal point is always retained.	a. The PICTURE must not contain more than one + or - and must not contain any A X S b. The PICTURE can contain Z's only if they follow a period and are the rightmost characters of the field excluding a report sign.	Can only be preceded by , . $ + - B 0
$	If one $, the insertion of the $ in the character position; if more than one in sequence, the equivalent of float dollar sign.	a. If one $, the PICTURE cannot contain A X b. If floating $, the PICTURE cannot contain more than one + or - must not contain A X or * and can contain Z's only if they follow a period and are the rightmost characters excluding a report sign.	a. Must be the leftmost character if a single $ is given. b. If floating, must be the leftmost receiving character positions and can only be preceded by the insertion characters . , - + B 0
, (comma)	Insertion of a comma in the character position so long as a significant digit precedes it; if the preceding character position is blanked or replaced by editing the comma position is blanked or replaced.	The PICTURE must not contain A X S	none
. (period)	Insertion of an actual decimal point unless the entire item consists of spaces because of receiving a zero value.	The PICTURE must not contain A X P V S	none
B	Insertion of a blank (i.e., a space) in the character position.	The PICTURE must not contain S	none
CR	Insertion of the two characters CR if the data item has a negative value; otherwise, two blanks or spaces are inserted.	The PICTURE must not contain any of the characters A X + - S DB	Must be the rightmost characters.

PICTURE CHARACTER	MEANING	RESTRICTIONS	REQUIRED PICTURE POSITION
DB	Insertion of the two characters DB if the data item has a negative value; otherwise, two blanks or spaces are inserted.	The PICTURE must not contain any of the characters A X + - S CR	Must be the rightmost characters.
0	Insertion (or presence) of the character zero in the character position.	a. In a PICTURE containing only S V P 9 0 characters, the 0's count as decimal places in the sending item; as insertion characters in a receiving item. b. In all other cases, 0's in a PICTURE are insertion characters.	none
+	Insertion of a plus sign in the character position if the data item has a positive value, or a minus sign if a negative value; when more than one in sequence, represents a floating sign with replacement of leading 0's, commas, and B's by spaces (the equivalent of BLANK WHEN ZERO).	a. When only one + is used, it must be either the rightmost or the leftmost character; the PICTURE must not contain A X - S CR and/or DB and the + can be preceded by P only if rightmost. b. When floating, the PICTURE must not contain A X - S CR DB * and no more than one $; it can contain Z's only if they follow a period and are the rightmost characters.	a. If single must be leftmost or rightmost character in the PICTURE. b. If floating it can be preceded by any insertion character except -.
-	Insertion of a minus sign in the character position if the data item has a negative value, or a blank if of a positive value; when more than one in sequence, represents a floating sign with replacement of leading 0's, commas, and B's by spaces (the equivalent of BLANK WHEN ZERO).	a. When only one - is used, it must be either the rightmost or the leftmost character; the PICTURE must not contain A X + S CR and/or DB and the - can be preceded by P only if rightmost. b. When floating, the PICTURE must not	a. If single must be leftmost or rightmost character in the PICTURE. b. If floating it can be preceded by any insertion character except +.

PICTURE CHARACTER	MEANING	RESTRICTIONS	REQUIRED PICTURE POSITION

PICTURE CHARACTER	MEANING	RESTRICTIONS	REQUIRED PICTURE POSITION
Z	Suppression of leading zeros and commas; if both to the right and the left of an assumed or actual decimal point (V or .), the equivalent of BLANK WHEN ZERO.	a. If following a period (.) the period can be preceded by any other floating character. b. If not following a period, the PICTURE can contain only one $ + or - and cannot contain any * A X S	Can only be preceded by , . V $ + - B 0

B.12. POINT LOCATION

FUNCTION: To define as assumed decimal point.

FORMAT:

(data-name-1...) [; POINT LOCATION IS {LEFT / RIGHT} integer-4 PLACES]

TECHNICAL NOTES:

1. When the PICTURE clause is not given, the definition of numeric items having non-integral values requires the POINT LOCATION clause.
2. An actual decimal point (i.e., the character ".") can not be defined through the use of this clause. Actual decimal points must be shown in a PICTURE clause.
3. Integer-4 must be a numeric literal with an integral value in the range: 1 through 32. The point is located integer-4 positions to the left or right of the least significant position of the field.
4. The SIZE of the data-item being described must not be greater than 63 characters.
5. This clause can be used only at an elementary item level.

RANGE

B.13. RANGE

FUNCTION: To specify the potential range of the value of an item.

FORMAT:

(data-name-1 ...) [; RANGE IS literal-1 THRU literal-2]

TECHNICAL NOTES:

1. For NUMERIC items literal-1 and literal-2 represent the respective minimum and maximum values of the item. Literal-2 must not contain more digits than are specified in the SIZE clause. If literal-2 requires less area in the Internal Format than the size specified, only the storage area required to represent literal-2 is allowed for. The RANGE of a NUMERIC item can be useful in handling the intermediate results of a complex computation.
2. For non-NUMERIC items, each character of literal-1 and literal-2 represents the respective minimum and maximum values of the corresponding character position in the item.
3. RANGE can be written only at the elementary item level.
4. Literal-1 and literal-2 can be Figurative Constants.
5. The RANGE clause is for documentary purposes only.

B.14. REDEFINES

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

FORMAT:

(data-name-1...) [; REDEFINES data-name-2]

TECHNICAL NOTES:

1. The REDEFINES clause, when used, must immediately follow the level-number data-name-1 clause.
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 1, 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. Neither data-name-1 nor data-name-2 can be qualified.
10. A REDEFINES clause cannot be subordinate to an OCCURS statement or appear within the same record description of an OCCURS statement.

SIGNED

B.15. SIGNED

FUNCTION: To specify the operational sign of an elementary item.

FORMAT:

(data-name-1...) [; SIGNED]

TECHNICAL NOTES:

1. This clause can be used only at the elementary item level.
2. An item which contains an operational sign must be NUMERIC.
3. SIGNED indicates the use of a standard operational sign. (This can also be indicated by the use of an "S" in the PICTURE.)
4. When a standard operational sign is indicated, the size, position and handling of the sign need not be shown. A standard operational sign is not considered in determining the SIZE.
5. An item which contains any editing symbols other than 0 cannot have an operational sign.
6. If a PICTURE is given, this clause is ignored.

SIZE

B.16. SIZE

FUNCTION: To specify the size of an item in terms of the number of standard, data-format characters.

FORMAT:

Option 1:

```
(data-name-1 ...)  [ SIZE IS integer-1  { CHARACTERS }  ]
                                        { DIGITS     }
```

Option 2:

```
(data-name-1 ...)  [ SIZE IS integer-1  TO integer-1a { CHARACTERS }  ]
                                                      { DIGITS     }
                   [ DEPENDING ON data-name-3 ]
```

TECNICAL NOTES:

1. The size of an item should be specified at the elementary item level by means of a SIZE clause or a PICTURE. A PICTURE and a SIZE clause need not both be given. If both are given, the SIZE clause is ignored. Use of the SIZE clause at any level other than the elementary item level is optional. The actual size of a group is the sum of the sizes of the elementary items comprising the group. A SIZE clause at a group level is useful only as documentation.

   If a group contains an item whose record description contains the integer-1 option of the OCCURS clause, the size of the generated table is determined by the product of the size of the table element and the number of occurrences.

2. Integer-1 represents the exact number of characters, excluding operational signs. If integer-1 is zero, the data being described will not necessarily be present at object time.

3. Integer-1 and integer-1a must be positive numeric literals and cannot contain a decimal point. When both are used, integer-1 must be less than integer-1a. The data description of data-name must be such that it defines a positive numeric elementary item with no character positions to the right of the assumed decimal point.

4. The DEPENDING ON option is only significant for 01 level FD entries that use the RECORDING MODE IS BCD clause and refer to blocked, variable-size records on input tape files. The contents of data-name-3 are used by the object program I/O to unblock the input records. Data-name-3 must reference a three-character, numeric field (see Section VII). The DEPENDING ON option is ignored in all other instances.

SYNCHRONIZED

B. 17. SYNCHRONIZED

FUNCTION: To describe the positioning of an elementary item within a computer word or words.

FORMAT:

(data-name-1) [; SYNCHRONIZED { LEFT / RIGHT }]

TECHNICAL NOTES:

1. This clause is for documentary purposes only, because the Series 200 processors are "character" not "word" machines.

2. This clause can be applied only to an elementary item.

B.18. USAGE

FUNCTION: To specify the dominant use of a data item.

FORMAT:

```
(data-name-1...) [ ; USAGE IS { DISPLAY          } ]
                               { DISPLAY-1        }
                               { DISPLAY-2        }
                               { COMPUTATIONAL    }
                               { COMPUTATIONAL-1  }
                               { COMP             }
                               { COMP-1           }
```

TECHNICAL NOTES:

1. The USAGE clause can be written at any level but is usually used for documentary purposes only, since all characters are stored in standard data format (six bits per character) within the Series 200.
2. DISPLAY and DISPLAY-1 are identical. All items are considered to be DISPLAY(-1) unless otherwise defined.
3. When the RECORDING MODE IS BCD clause has been specified, all items within the file so described are considered DISPLAY-2 items unless otherwise defined. (See page VII-20.)

VALUE

B.19. VALUE

FUNCTION: To define the values of constants or the initial values of working-storage items.

FORMAT:

```
(data-name-1...) [;{VALUE IS / VALUES ARE} literal-3 [THRU literal-4]
  [, literal-5 [THRU literal-6] ...]]
```

TECHNICAL NOTES:

1. The VALUE clause can be used in a working-storage or constant entry.
2. A VALUE clause has no meaning when applied to an input or an output file record.
3. A Figurative Constant can be used in place of literal-3. (The Figurative Constant can be in the form of ALL any Literal.)
4. The VALUE clause must not be stated in a Record Description entry which contains an OCCURS clause or in an entry which is subordinate to an entry containing an OCCURS clause.
5. 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 VALUES cannot be stated at the subordinate levels within the group.
6. When VALUE is not specified, the initial contents in the working-storage areas is zero.
7. The entries giving the new description of the storage area for a redefined item may not contain any VALUE clauses.
8. When the

   literal-3 THRU literal-4

   option is used, the second literal within the phrase must be larger than the first; i.e., literal-4 must be larger than literal-3. There must be at least two literals in a THRU entry.

documentary

B.20. ZERO SUPPRESS

FUNCTION: To permit suppression of non-significant zeros and commas.

For format and technical notes, see "Editing Clauses" (Section XI).

DATA DIVISION SECTION ENTRIES

File Section

The FILE SECTION contains a section header, File Description entries, and Record Description entries. Some of the information about the file may be in the COBOL library and therefore may not appear explicitly in the FILE Section. 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 32,768 (15 bit) locations. The difference between WORKING-STORAGE and FILE storage is that the former deals with computer memory requirements for the storage of intermediate data results, whereas the latter 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 non-contiguous 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.
          12
          77 data-name-1
                .
                .
                .
          77 data-name-n
          01 data-name-2
             02 data-name-3
                   .
                   .
                   .
          01 data-name-4
             02 data-name-5
               03 data-name-n
```

NOT BROKEN DOWN.

Broken Down.

77s begin in column 12.

Items in working storage which bear no relationship to one another need not be grouped into records, provided they do not need to be further subdivided. Instead, they are classified and defined as non-contiguous items. Each of these items is defined in a separate Record Description entry which begins with the special level number 77.

The following Record Description clauses are required in each entry:

1. Level-number
2. Data-name
3. PICTURE or SIZE

The OCCURS clause is not meaningful for non-contiguous items and causes an error at compilation time if used. Other Record Description clauses are optional and can be used to complete the description of the item, if necessary.

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 including REDEFINES, and OCCURS.

The initial value of any item in the WORKING-STORAGE SECTION may be specified by using the VALUE clause of the Record Description. (See VALUE page V-45.) 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 SIZE clause (or PICTURE) of the associated data entry, but it cannot be greater.

Constant Section

The concept of literals and figuratives enables the user to specify the value of a constant by writing its actual value (or a figurative representation of that value). It is often desirable to name this value and then refer to it by its name. For example:

> 6% (.06) may be named as INTEREST-RATE and then referred to by its name (INTEREST-RATE) instead of its value (.06).

Constant storage is, therefore, that part of computer memory which is set aside to save named constants for use in a given program. The CONSTANT SECTION must not use more than 32,768 (15 bits) locations.

The CONSTANT SECTION is organized in exactly the same way as the WORKING-STORAGE SECTION, beginning with a section header, followed by Record Description entries for non-contiguous constants, and then by Record Description entries for contiguous constant records

and their subordinate entries, in that order. The skeletal format for the CONSTANT SECTION is as follows:

```
CONSTANT SECTION.
    77 data-name-1
       .
       .
       .
    77 data-name-n
    01 data-name-2
       02 data-name-3
          .
          .
          .
    01 data-name-4
       02 data-name-5
          03 data-name-6
          .
          .
          .
```

Constants which bear no relationship to one another need not be grouped into records provided they do not need to be further subdivided. Instead they are classified and defined as non-contiguous constants. Each of these constants is defined in a separate Record Description entry which begins with the special level number 77.

The following Record Description clauses are required in each entry:

1. Level Number
2. Data-name
3. PICTURE or SIZE
4. VALUE

Other Record Description clauses are optional and can be used to complete the description of the constant when necessary.

Named constants in the CONSTANT SECTION which bear a definite relationship to one another must be grouped into records according to the rules for formation of Record Descriptions (see page V-14). All Record Description clauses can be used in a Constant Record Description, including REDEFINES, OCCURS, and COPY. Each constant storage record-name (01 level) must be unique since it cannot be qualified by a file-name or section-name. Subordinate data-names need not be unique if they can be made unique by qualification.

In the definition of constants, the VALUE clause is required and can only be specified for data items containing homogeneous characters (i. e. , characters having the same USAGE). VALUE, therefore, cannot be specified at levels which contain characters having different

USAGEs. All the rules for the expression of literals and figurative constants apply (see page I-3). The size of a literal used to specify the value of a constant can be equal to or less than the size specified in the SIZE clause of the associated data entry, but it cannot be greater.

Tables of constants, to be referenced by means of subscripting, are defined in one of the following ways:

1. The table can be described as a record by a set of contiguous Record Description entries, each of which specifies the VALUE of an element, or part of an element, of the table. In defining the record and its elements, the Record Description clauses may be used to complete the definition where required. The hierarchical structure of the table is then shown by use of the REDEFINES entry and its associated subordinate entries. The subordinate entries following the REDEFINES entry which are repeated due to OCCURS clauses must not contain VALUE clauses.
2. When the elements of the table do not require separate handling, the VALUE of the entire table can be given in the entry defining the entire table. The lower-level entries show the hierarchical structure of the table. Lower-level entries must not contain VALUE clauses.

Condition-Name Entry

See Section XI for a complete description of condition-name entries.

SECTION VI
THE PROCEDURE DIVISION

General

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, which in turn can be combined to form sections under section-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 or a section-name not used in any other division.
4. A figurative constant.

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 either:

1. DECLARATIVES (followed by the source program's USE verb statements grouped as sections),

or 2. A paragraph-name or section-name.

A section-name must always be immediately followed by a paragraph-name (except in the DECLARATIVES section, see USE). Note that if a DECLARATIVE section is used, there must be at least one more named section (which must follow the DECLARATIVE SECTION).

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 a section, or a group of successive sections 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, each has in common that the verb must be the first word of the statement. For COBOL purposes, the word IF is regarded as a verb since coding is generated in response to its source

program occurrence. The verb statements can be categorized as being one of three types, an imperative statement, a conditional statement, or a compiler-directing statement. The verbs are categorized as follows:

A. Imperative

1. Arithmetic

These verbs are:

ADD
SUBTRACT
MULTIPLY
DIVIDE

2. Sequence Control (Branching)

These verbs are:

ALTER
CALL
GO TO
PERFORM

3. Data Movement

These verbs are:

EXAMINE
MOVE

4. Ending

This verb is:

STOP

5. Input/Output

These verbs are:

ACCEPT
CLOSE
DISPLAY
LOAD
OPEN
READ
WRITE

B. Conditional

This verb is:

IF

C. Compiler-Directing

These verbs are:

EXIT
INCLUDE
NOTE
USE

The verbs are discussed individually in alphabetic order under PROCEDURE DIVISION VERB FORMATS below.

Statements

An imperative statement consists of either a verb (excluding IF, USE, 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 one of the following two forms:

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

   or

2. Imperative statement followed by a conditional statement.

In form 2, the imperative statement must not end with a GO or a STOP RUN statement. When the ON SIZE ERROR option is used with an arithmetic verb, the verb, its operands, and the option are considered a conditional statement of the second form. Note that since an AT END statement exists explicitly for every READ, READ statements are always conditionals of the second form. In form 1, statement-1 and statement-2 can be either imperative or conditional, and can, in turn, contain conditional statements. The conditions within the conditional statement are considered to be "nested." NEXT SENTENCE can be substituted for the statement-1 or statement-2. If substituted for statement-2, the whole clause OTHERWISE (or ELSE) NEXT SENTENCE can be omitted.

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 is called a procedural sentence.

1. 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 THEN GO TO START.

2. Conditional Sentences

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

   EXAMPLE: IF X EQUALS Y THEN MOVE A TO B; OTHERWISE MOVE C TO D.

If the phrase "OTHERWISE NEXT SENTENCE" immediately precedes the period, then the phrase "OTHERWISE NEXT SENTENCE" can be eliminated. This rule can then be applied to the resulting sentence.

3. Compiler-Directing Sentences

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

EXIT.

4. 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 or the section referenced by the GO TO statement.

a. Imperative Sentences

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).

b. Conditional Sentences

IF condition { statement-1 / NEXT SENTENCE } { OTHERWISE / ELSE } { statement-2 / NEXT SENTENCE }.

In the conditional sentence form above, the condition is an expression which is true or false. If the condition is true, then statement-1 is executed and control is transferred to the next sentence. If the condition is false, statement-2 is executed and control is passed to the next sentence. The only exception to this is if statement-1 or statement-2, whichever is applicable, is an unconditional GO TO.

If statement-1 is conditional, the conditional statement must be the last, or the only, statement comprising statement-1. For example, in a conditional sentence having the form:

IF condition-1 imperative-statement-1 IF condition-2 statement-3 OTHERWISE statement-4 OTHERWISE statement-2.

If condition-1 is true, imperative-statement-1 is executed. Then, if condition-2 is true, statement-3 is executed and control is transferred to the next sentence. If condition-2 is false, statement-4 is executed and control is transferred to the next sentence. If condition-1 is false, statement-2 is executed and control is transferred to the next sentence.

Note that statement-3 can, in turn, be either conditional or imperative. If conditional, it can itself contain conditional statements in arbitrary depth. In the same manner, statement-4 (and statement-2) can be either imperative or conditional.

c. Compiler-Directing Sentences

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.

5. 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 rule of 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 procedure can reference another by naming the procedure to be referenced. In this way, the sequence in which the object program is to be executed can be varied simply by transferring to a named procedure. The name consists of a word followed by a period, and the name precedes the procedure it names.

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 procedure 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 followed by the name of the procedure 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 in accordance with the rules of the PROCEDURE DIVISION, and the requirements of the reference format, the source programmer begins a paragraph with a name. The smallest grouping of the PROCEDURE DIVISION which is named is a paragraph.

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.

Sections

A section consists of one or more successive paragraphs and, when designated, must be named. The section-name is followed by the word SECTION, a priority number which is optional and a period. If the section is the DECLARATIVES section, then the declarative sentence (i.e., USE) follows the section header. The section-name applies to all paragraphs following it until

another section-name is found. It is not required that a program be broken into sections, but the occurrence of a DECLARATIVES section requires that there be at least one more named section and that this named section immediately follow the DECLARATIVES section.

When a priority number is used, the following ground-rules apply:

1. All sections which have the same priority number constitute a program segment with that priority. All sections with no priority number are assumed to belong to segment 0. A segment is considered called whenever a GO TO or a PERFORM statement references a procedure-name within that segment.
2. Segments with a priority number in the range from 0 to the SEGMENT LIMIT minus 1 are part of the fixed portion (resident portion) of the object program. This means that these segments are loaded into memory only once, at the beginning of the object run, and are never overlaid.
3. Segments with a priority number in the range from segment limit to 49 are part of the nonresident portion of the object program. They are brought into memory whenever needed. Any ALTERable switches (paragraphs consisting solely of a GO TO statement) in these segments retain their last set value regardless of when the segment is brought into memory.
4. Segments with a priority number of 50 or greater are brought into memory each time one is called regardless of the previous calling. Any ALTERable switches in these segments are reinitialized whenever the segment is called.
5. Sections in the Declaratives must not contain priority numbers in their section headings. All other sections may have a priority number.

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 purposes of this discussion.

In the discussion of each verb, there are two sections:

Object Coding Produced — in which, so far as possible, information is given as to the coding generated in response to the various possible ways of constructing the verb statement.

General — in which the rules governing the construction and use of the verb statement are given.

In connection with the Object Coding Produced section for each verb, the programmer should keep in mind that the appearance of subscripted data-names in PROCEDURE DIVISION paragraphs results in additional object code generation. In most cases, an 11-character subroutine calling sequence is generated at the first appearance of each unique subscript within a paragraph.

ACCEPT

A.1. ACCEPT

FUNCTION: To receive low-volume data from a peripheral device. (Card reader [unless Console available])

VERB FORMAT:

ACCEPT data-name [FROM mnemonic-name]

TECHNICAL NOTES:

1. Object Coding Produced

   A 10-character subroutine calling sequence is generated.

2. General

   a. The ACCEPT device may be either the card reader or the console typewriter. If the card reader has been designated and data-name field exceeds 80 characters (160 if the card reader is in transcription mode) the card reader truncates data-name at 80 (or 160) characters. If the console typewriter has been designated, data is accepted until the end of the operand field is reached.

   b. If the

      FROM mnemonic-name

      option is specified, mnemonic-name must have been used in the SPECIAL-NAMES paragraph.

ADD

A. 2. ADD

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

VERB FORMAT:

Option 1:

ADD {data-name-1 / literal-1} [, {data-name-2 / literal-2 ...}] [TO] [{data-name-m / literal-n} GIVING]
data-name-n [ROUNDED] [; ON SIZE ERROR any imperative statement]

Option 2:

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

TECHNICAL NOTES:

1. Object Coding Produced

   a. The basic coding generated is 7 characters per operand (or 15 characters per redefined operand), excluding the receiving operand when the GIVING option is not used.

   b. A common point location is determined equal to the largest of the point locations in all the operands. Then for each operand with a point location not the same as the common point location, coding is produced to move the field to a temporary field. This coding is that generated for a Numeric-move (see MOVE verb) except that instructions whose only function is to normalize the sign are suppressed. If the SIZE ERROR or the ROUNDED clause or both are present, or if the receiving field is edited, a move to the receiving field is generated even when the point location of the receiving field is the same as the common point location.

   c. When the SIZE ERROR clause is present, 10 more characters are produced; and if the receiving field is defined with fewer integer places than the largest of the integer places in the addends, 12 more characters are generated.

   d. When the ROUNDED clause is present, 7 more characters are produced.

eg. Hart, or add it somewhere else; because the addition is made in a work area & if oversize, does not get put into the receiving field, so it must be taken care of elsewhere.

2. General

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

b. Only numeric fields and numeric literals can be used as operands in ADD verb statements. The use of an alphabetic field results in a fatal diagnostic. The use of an alphanumeric field results in a warning diagnostic. The only figurative constant that can be used is ZERO(E)(S).

c. Of the operands used, the rightmost must be a data-name. Any data-name used can reference an elementary item only. Neither the addend nor the augend can contain editing symbols. ~~The rightmost, and only the rightmost, data-name can contain editing symbols~~. When it does contain editing symbols, it must be a receiving field not used in the computation. (i.e. using "GIVING" option).

d. 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. However, not more than 13 basic operands

It should be noted, however, that under certain conditions right truncation of integer places and left truncation of decimal places is possible. For example, given the sum 645821Λ and a receiving field with a PICTURE of 9999PPPV, decimal point alignment would be:

```
0645 8 2 1Λ
9999PPPV
```

and the digits stored would be 0645. Similarly, given the sum Λ8259723 and a receiving field with a PICTURE of VPPPP9999, decimal point alignment would result in:

```
Λ 82 5 97230
VPPPP9999
```

and the digits stored would be 7230.

i. SIZE ERROR and ROUNDED options are provided for those cases where either or both types of truncation can take place.

1) When the SIZE ERROR option is specified, a test is made at object time to see if left digit truncation will occur when the sum is stored in the receiving field. If left truncation will occur, the sum is not stored. Instead, the "any imperative statement" associated with the SIZE ERROR option is performed.

2) When the ROUNDED option is specified, a test is made at compilation time to see if right digit truncation will occur when the sum is stored in the receiving field. If right truncation will occur, the least significant digit of those digits that can be stored is increased by 1 when the most significant digit of those that will be truncated is in the range 5 through 9.

3) If the SIZE ERROR option is not specified and a size error condition occurs, the value of the results are unpredictable.

j. Editing symbols are permitted in the receiving field for the sum so long as that field is not used as an augend or an addend.

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

ADD A TO B (another way of stating ADD A B)

is the logical equivalent of

A + B = B.

Again,

ADD A, B TO C (another way of stating ADD A, B, C)

is the equivalent of

A + B + C = C.

However,

ADD A TO B GIVING C (another way of stating ADD A, B GIVING C)

is equivalent to

A + B = C

While

ADD A, B TO C GIVING D (another way of stating ADD A, B, C GIVING D)

is equivalent to

A + B + C = D.

That is, if the word TO is used in an ADD verb statement and it is not followed by a

data-name-m GIVING data-name-n

clause, the data-name following TO is used in the arithmetic process and the sum is stored in data-name. When

... TO data-name-m GIVING data-name-n

is given, data-name-m is used in the arithmetic process, but its value is left unchanged. The sum is stored in data-name-n (which is not used in the arithmetic process).

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

m. For a complete description of ADD CORRESPONDING, see "Language Preprocessor" (Section XI).

A.3. ALTER

FUNCTION: To modify a predetermined sequence of operations.

VERB FORMAT:

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

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

TECHNICAL NOTES:

1. Object Coding Produced

   Seven in-line characters and three out-of-line characters are generated. The out-of-line characters are a constant which defines the destination (generated duplicate constants are eliminated).

2. General

   a. Procedure-name-1, procedure-name-3,..., must be names of paragraphs which contain a single sentence consisting of:

      GO TO procedure-name.

   b. A GO statement in a SECTION whose priority is 50 or greater cannot be referenced by an ALTER in a SECTION having a different priority.

CALL

A.3.1 CALL

FUNCTION: To execute an Easycoder program previously loaded.

VERB FORMAT:

$$\underline{\text{CALL}}\ \text{program-name}\ \left[\ \underline{\text{USING}}\ \left\{\begin{matrix}\text{data-name-1}\\ \text{literal-1}\end{matrix}\right\}\ \left[,\ \left\{\begin{matrix}\text{data-name-2}\\ \text{literal-2}\end{matrix}\right\}\ \ldots\ \right]\ \right]$$

TECHNICAL NOTES:

1. Object Coding Produced

   a. B (LOC)
   DSA of data-name-1 or 1 character DC
   DSA of data-name-2 or 1 character DC
   .
   .
   .

   b. LOC is an address in the object input/output routine. When the LOAD verb is executed, the starting address of the program is stored in LOC.

   c. The last DSA, DC, or the right character of the branch instruction contains a right item-mark.

2. General

   a. Program-name must have been previously loaded by a LOAD statement.

   b. Data-name-1 and data-name-2 must not be subscripted.

   c. Literal-1 and literal-2 must not be more than one character.

   d. The Easycoder program is activated at the location specified in its END or EX card. This information has been previously recorded during execution of the corresponding LOAD statement.

3. Suggested Linkage Instructions Within the Easycoder Program

   A recommended format for the first instructions within the user's Easycoder program is shown below. These instructions provide communications between the COBOL calling program and the Easycoder program.

```
                SCR         EXIT+3, BREG
                EXM         (EXIT+1), PARAMS, LRID
                SCR         EXIT+3, AREG
                 .
                 .
                 .
EXIT            B           000
PARAMS          DSA         000
                DSA         000
                DC          @0@
LRID            CEQU        #1C51
BREG            CEQU        #1C70
AREG            CEQU        #1C67
```

   The user references data-name in his COBOL object program using indirect addressing via PARAMS.

4. External Programming Tips

a. Data-names in a COBOL object program have left word marks, right item-marks, both, or neither. The rules governing these situations are too numerous and complex to be of practical value to the user. A series of OBSERVATION diagnostics are generated and refer to the data-names in the CALL statement.

If a diagnostic does not appear, the data-name has a left word-mark and no other punctuation. All other punctuation produces a self-explanatory diagnostic.

b. Normally, the user should not change punctuation. If the user changes punctuation on a data-name, he must be careful to restore it exactly as it was prior to the change.

c. If the user chooses to use index registers, the index registers should be saved and restored.

d. If the user chooses to perform input/output functions, the following should be performed:

(1) Reserve read/write channels through the use of an RWCS instruction, and

(2) Test the read/write channel for busy; when it is free, save and restore the ending counter.

e. The user should not manipulate devices referenced by the object input/output package.

f. If the user wishes to use a CSM instruction, the co-sequence register should be saved and restored.

g. The external program is activated in the same address mode (3 or 4 characters) operative in the COBOL object program.

h. The external program must return control to the COBOL object program in the same address mode (3 or 4 character) as was operative when it was activated.

i. The user should not reference the Loader-Monitor within his external program.

CLOSE

A.4. CLOSE

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

VERB FORMAT:

$$\underline{\text{CLOSE}}\ \text{file-name-1}\left[\left\{\begin{matrix}\underline{\text{WITH}}\\ \underline{\text{REEL}}\end{matrix}\ \left\{\begin{matrix}\underline{\text{NO REWIND}}\\ \underline{\text{LOCK}}\end{matrix}\right\}\right\}\right]\ \left[,\ \text{file-name-2}\ldots\right]$$

TECHNICAL NOTES:

1. Object Coding Produced

   An eight-character subroutine calling sequence is generated.

2. General

   a. A file must be OPENed before it can be CLOSEd. An OPEN is valid for the file to which it is applied until a CLOSE is given. Only one CLOSE (without REEL) can be given for a file for each time it has been OPENed.

   b. The occurrence of either

CLOSE Statement	File Opened on a Peripheral Device	Multiple file reel	File Opened on Tape
CLOSE file-name	note C	notes C and B	notes C, G, and E
CLOSE file-name WITH LOCK	notes C and D	notes C, B, and D	notes C, G, D, and E
CLOSE file-name WITH NO REWIND	note C	notes C and A	notes C and A
CLOSE file-name REEL	note H	ILLEGAL	notes F and G

Notes:

A. The current reel is left in its current position.

B. The entire reel is rewound to the beginning of the reel, regardless of the position of the file on the reel.

DISPLAY

A.5. DISPLAY

FUNCTION: To display low-volume data.

VERB FORMAT:

$$\underline{\text{DISPLAY}} \left\{ \begin{matrix} \text{literal-1} \\ \text{data-name-1} \end{matrix} \right\} \left[, \left\{ \begin{matrix} \text{literal-2} \\ \text{data-name-2} \end{matrix} \right\} \ \ldots \right] \left[\underline{\text{UPON}} \ \text{mnemonic-name} \right]$$

TECHNICAL NOTES:

1. Object Coding Produced

   The basic coding generated for this verb is a 10-character subroutine calling sequence. If there is more than one operand, coding is generated in front of the calling sequence to move each operand to adjacent temporary fields. (This coding is described under the MOVE verb, Non-numeric MOVE.)

2. General

   a. The DISPLAY device may be either the printer or the console typewriter. All fields specified in the DISPLAY statement are placed side-by-side in their given order. If the printer is used for display and the sum of the sizes of all the fields is greater than the size of the printer line, the rightmost data is lost. If the console typewriter is used for display, all fields are typed.

   b. If the

      UPON mnemonic-name

      option is used, mnemonic-name must have been specified in the SPECIAL-NAMES paragraph.

DIVIDE

A.6. DIVIDE

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

VERB FORMAT:

```
DIVIDE { data-name-1 } INTO [ { data-name-2 } GIVING ] data-name-3 [ROUNDED]
       { literal-1   }        { literal-2   }

[; ON SIZE ERROR any imperative statement ]
```

TECHNICAL NOTES:

1. Object Coding Produced

   a. The basic coding generated for this verb is a 14-character subroutine calling sequence. Eight more characters are produced for each operand which is redefined. Coding as described under the MOVE verb, Numeric Move, is generated to move the dividend to a temporary field.

      If the ROUNDED clause or the SIZE ERROR clause or both are present, or if the result field contains editing symbols, a second move is generated to move the quotient to the result field.

   b. If ROUNDED is specified, 7 more characters are generated.

   c. If SIZE ERROR is specified, 14 more characters are generated.

2. General

   a. Each DIVIDE verb statement must contain at least two operands (i.e., a dividend and a divisor).

   b. Only numeric fields and numeric literals can be used as operands in a DIVIDE verb statement. The use of an alphabetic field results in a fatal diagnostic. The use of an alphanumeric field results in a warning diagnostic.

   c. Any operand which is also to serve as a receiving field must be a data-name. No field containing editing symbols can be used as a dividend or a divisor.

   d. Each operand can have an operational sign and an implied decimal point. If no decimal point is indicated, it is assumed to be to the right of the least significant digit of the operand. There is no practical limit to the number of digits in a DIVIDE statement operand, except that the field which is to receive the quotient cannot exceed 63 characters.

   e. Division by zero constitutes a special type of "size error" and the rules specified under g., below, apply.

   f. Truncation of left and/or right 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, the result is MOVEd to the receiving field according to the rules of numeric MOVEs.

g. SIZE ERROR and ROUNDED options are provided for those cases where truncation can take place.

   1) When the SIZE ERROR option is specified, a test is made to see if left digit truncation will occur when the quotient is to be stored in the receiving field. If left truncation will occur, the quotient is not stored. Instead, the "any imperative statement" associated with the SIZE ERROR option is performed.

   2) 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.

   3) If the SIZE ERROR option is not specified and a size error condition occurs, the value of the result is unpredictable.

h. Editing symbols are permitted in the receiving field for a quotient so long as that field is not used as a dividend.

EXAMINE

A.7. EXAMINE

FUNCTION: To count and/or replace the number of occurrences of a given character in a data item.

VERB FORMAT:

```
EXAMINE data-name { TALLYING { ALL       }                literal-1 [REPLACING BY literal-2] }
                  {          { LEADING   }                                                   }
                  {          { UNTIL FIRST }                                                 }
                  { REPLACING { ALL       }  literal-3 BY literal-4                         }
                  {           { LEADING   }                                                 }
                  {           { [UNTIL] FIRST }                                             }
```

TECHNICAL NOTES:

1. Object Coding Produced

   The basic coding generated for this verb is a 12-character subroutine calling sequence. If the field being EXAMINEd is SIGNED, 24 more characters are generated.

2. General

   a. Each of the literals in the format must be either a single-character non-numeric literal or a figurative constant.

   i.e. if a numeral must be enclosed in quotes.

   b. The EXAMINATION of data-name proceeds from left to right.

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

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

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

      3) Characters not equal to literal-1 but not following an occurrence of literal-1 when the UNTIL FIRST option is used.

   d. When either REPLACING option is used:

      1) When the ALL option is used, the second named literal is substituted for each occurrence of the first named literal.

      2) When the LEADING option is used, the substitution of the second named literal for the first named literal terminates when a character other than the first named literal is encountered or the rightmost character of the data item has been examined.

      3) When the UNTIL FIRST option is used, the substitution of the second named literal for data item characters terminates as soon as the first named literal is encountered or the rightmost character of the data item has been processed.

      4) When the FIRST option is used, the first occurrence of the first named literal is replaced by the second named literal.

e. The count kept by the TALLYING option is stored in a data field generated by the compiler. The name of this field is TALLY, and its characteristics are those of a field corresponding to the PICTURE S9(5). TALLY can be referenced in the source program by use of the fixed data-name TALLY.

Programming

The count kept by the TALLYING option can be referenced by use of the data-name TALLY in the source program. TALLY can also be used for storage, or set to a value by the programmer, remembering that the execution of

EXAMINE data-name TALLYING

will destroy the previous content of TALLY.

"Tally" is only referenceable in the instance where the Examine verb is used in the program. (Also you could use this Tally area for other tally's, but be careful.)

A.8. EXIT

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

VERB FORMAT:

EXIT.

TECHNICAL NOTES:

1. Object Coding Produced

   The occurrence of EXIT in the source program causes no generation of object code.

2. General

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

      LOOP-OUT. EXIT.

   b. 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.

   c. 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.

A.9. GO TO

FUNCTION: To depart from the normal sequence of procedures.

VERB FORMAT:

GO TO procedure-name

TECHNICAL NOTES:

1. Object Coding Produced

   From three to eleven characters are generated for this verb. The exact number depends on whether procedure-name is in a nonresident segment and whether three- or four-character addressing is being generated.

2. General

   a. If the GO TO procedure-name statement is to be ALTERed, then

      1) The GO TO procedure-name statement must immediately succeed a paragraph-name.

      2) The paragraph containing the GO TO procedure-name statement can contain only the GO TO procedure-name statement.

   The paragraph-name assigned to the GO TO procedure-name statement is referred to by the ALTER verb to modify the sequence of the program.

IF

A.10. IF

FUNCTION: To provide for the testing of stated conditions 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 conditions determines the subsequent operations to be performed.

VERB FORMAT:

CONDITIONAL STATEMENT:

```
IF { simple-condition   } { statement-1     } { OTHERWISE } { statement-2     }
   { compound-condition } { NEXT SENTENCE } { ELSE      } { NEXT SENTENCE }
```

1. Simple-Condition

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

a.

```
{ data-name-1 }   { IS [NOT] GREATER THAN      }   { data-name-2 }
{ literal-1   }   { IS [NOT] LESS THAN         }   { literal-2   }
                  { IS [NOT] EQUAL TO          }
                  { IS UNEQUAL TO              }
                  { EQUALS                     }
                  { IS [NOT] =                 }
                  { EXCEEDS                    }
```

(Note: literal-1 and literal-2 cannot both be used in the same statement.)

A simple condition of this type is called a relation test.

b.

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

(i.e., data-name content is [not] greater than zero, data-name content is [not] less than zero, or data-name content is [not] equal to zero.[1])

c.

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

(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 a space or spaces).

d.

```
[NOT] 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.

e. [NOT] condition-name

See Section XI.

2. Compound-Condition

The basic format for a compound-condition clause is

simple-condition-1 { AND / OR } simple-condition-2 [{ AND / OR } simple-condition-3 { AND / OR } ... simple-condition-n]

TECHNICAL NOTES:

1. Object Coding Produced

a. The following discussion of coding generated in response to conditional statements applies only to simple conditions. The presence of compound or abbreviated conditional statements in the source program has no effect on the coding generated. Such statements are treated as sequences of independent simple conditions.

Without knowing the precise structure of a given conditional statement and the context in which it appears, it is impossible in a summary discussion such as this to specify precisely how many characters would be generated during compilation. In many cases, the final generated coding can be four or even more characters shorter than that described below.

b. When the status of one of the breakpoint switches is tested, nine characters are generated.

c. When a field is tested for ALPHABETIC, a 17-character subroutine calling sequence is produced.

d. When a field is tested for NUMERIC, 16 characters are generated for unSIGNED single-character fields; 28 characters for SIGNED single-character fields; 41 characters for unSIGNED multi-character fields; and 64 for SIGNED multi-character fields.

e. When an unSIGNED field is tested for POSITIVE, NEGATIVE, or ZERO, or is compared against the figurative constant ZERO or ALL "0", 16 characters are generated. Eight more characters are produced if the field is redefined. When a SIGNED field is tested under the same conditions, 16 characters are generated plus 8 more characters if the field is redefined, plus 8 more characters if the words GREATER, LESS, or EXCEEDS are used.

f. When a field is tested for equality (or inequality) to a figurative constant other than ZERO or a single-character ALL literal, 28 characters are produced, plus 8 more if the field is redefined. However, if the field itself is a single character, only 12 characters are produced.

the items is shorter than the other, in effect, spaces are added to the low order end of that item until a field of the same length as the longer item has been constructed.

If either operand has USAGE DISPLAY-2 specified explicitly or implicitly, both operands are translated into IBM COMMERCIAL-COLLATE sequence before the above operations are applied. (See VII-20.)

b. A compound condition made up exclusively of relation tests, each of which has the same subject and the same relation, may be abbreviated by omitting all but the first subject and relation. For example, the compound condition

IF A = B AND A = C OR A = D

could be stated as

IF A = B AND C OR D

Furthermore, if all the connectors (AND or OR) in such a compound condition are the same, all except the last may be omitted or replaced by commas. For example, the compound condition

IF A = B AND A = C AND A = D

can be stated as

IF A = B, C AND D

As a further example, the compound condition

IF X IS LESS THAN Y AND X IS LESS THAN Z

may be abbreviated to

IF X IS LESS THAN Y AND Z

There may not be more than eight objects within a conditional statement. For example,

IF A=B, C, D, E, F, G, H, OR I (But seven is found to be the practical limit!)

Another possible abbreviation is described below.

When { data-name-1 / literal-1 } , the relation to be tested, and the logical connector (AND or OR) in each of the simple-conditions that make up the compound condition are the same, the data-name or literal, and the relation to be tested need only be written in the first simple-condition. The only necessary appearance of the logical connector is before the last { data-name-2 / literal-2 }

All other occurrences of the logical connector are either omitted or replaced by commas. For example:

IF A EQUALS B OR A EQUALS C OR A EQUALS D ... OR A EQUALS V imperative-statement

can be abbreviated,

IF A EQUALS B, C, D ... OR V ...

or

IF A EQUALS B C D ... OR V ...

c. Grouping by parentheses within compound-conditions can be used for ease of reading and to indicate the order of precedence of the relations to be established. For example,

IF A = B AND E IS GREATER THAN Y AND Z = D OR A = C and A = B imperative statement

can be shown in nested parentheses,

IF (A = B AND (A = C OR (Z = D AND E IS GREATER THAN Y)))...

Note that parentheses must be paired.

d. Compound-conditions containing three or more simple-conditions are evaluated:

1) The compound-condition is successively reduced by replacing each pair of simple-conditions connected by AND with another condition which is true only if both of the simple-conditions were true.

2) When all ANDs have been replaced by the above rule, what remains must be a set of conditions connected entirely or by ORs. This reduced compound-condition is then true if any of the conditions in it is true.

3) When parentheses are used in compound-conditions, the effect is to place a restriction on the order of reduction of conditions connected by AND. The restriction is that, starting with the innermost-paired parentheses, the set of conditions within the parentheses must be reduced to one condition (by successive applications of 1 and 2 above) before any reductions which cross parentheses can be performed.

e. At object-time, within any compound-condition, the simple conditions are evaluated from left to right. Those whose truth or falsity no longer has any pertinence to this execution of the coding are bypassed. Thus, given the compound-condition:

IF A EXCEEDS B OR C IS LESS THAN D AND E IS GREATER THAN F GO TO...

A would be tested against B. If A exceeds B, control would be sent to GO TO... . Only if A did not exceed B would the conditions following be tested.

3. Programmer

So far as possible, it is advisable that the important or crucial relations to be tested be in the beginning of a compound-condition. That is, in the case where a single test can determine that the true or false exit be taken, that relation should be written first. For example,

IF (A IS EQUAL TO B, C OR D) AND E IS UNEQUAL TO F...

should be written

IF E IS UNEQUAL TO F AND (A IS EQUAL TO B, C OR D)...

This is true even though parentheses are used. The object coding produced will be such that the innermost expressions within parentheses are logically evaluated first. However, the most efficient object coding is produced if the innermost parenthesized expressions are at the beginning of the conditional statement.

A. 11. INCLUDE

FUNCTION: To use library routines in a source program.

VERB FORMAT:

$$\left\{ \begin{array}{l} \text{section-name } \underline{\text{SECTION}}. \\ \text{paragraph-name}. \end{array} \right\} \ \underline{\text{INCLUDE}} \text{ procedure-name}.$$

TECHNICAL NOTES:

1. Object Coding Produced

   The object coding produced is dependent upon the source language lines INCLUDEd from the library. INCLUDE itself produces no object coding.

2. General

   a. The procedure-name refers to a COBOL procedure (library-routine) that has been stored in the library. The library-routine referenced by procedure-name consists of one or more COBOL paragraphs. The routine is copied during compilation and the result of the compilation is the same as if the routine were actually part of the source program.

   b. Each INCLUDE statement can reference only one procedure-name in the library.

   c. When the INCLUDE statement is preceded by a paragraph-name, the library entry must consist of a series of verbs and operands without the paragraph-name. When the INCLUDE statement is preceded by a section-name SECTION, the library entry must consist of a series of paragraphs without the section-name.

   d. When a paragraph is INCLUDEd, the INCLUDE verb statement must be the only entry in the paragraph.

   e. When a section is INCLUDEd, the INCLUDE verb statement must be the only entry in the section.

A.11.1 LOAD

LOAD

FUNCTION: To load a program into memory.

VERB FORMAT:

LOAD program-name-1 [, program-name-2...]

TECHNICAL NOTES:

1. Program-name is a maximum of eight characters. The rightmost two characters are assumed to represent a segment name and are right-justified. The remaining characters are left-justified and space-filled.

   Examples: ABLE01 is treated as ABLE△△01

   XYZ is treated as X△△△△△YZ

2. Program-name must be placed on the object run tape using the Update and Select program. (see Section X.)

3. When this statement is executed, the object-time input/output routine searches forward on the object run tape for program-name and loads it into memory.

4. NO SEGMENTATION must not be used when the LOAD verb is specified.

5. When more than one program-name is specified, the order of the program-names in the LOAD statement and the order of the programs on the object run tape must agree.

6. After all programs specified in the LOAD statement are loaded, the object run tape is positioned backwards until the segment containing the LOAD statement which specifies the program-name is found. The user must ensure that no duplicate object program-name occurs in the range of the object program and the external program.

A.12. MOVE

FUNCTION: To transfer data, in accordance with the rules of editing, to one or more data areas.

VERB FORMAT:

Option 1:

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

Option 2:

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

TECHNICAL NOTES

1. Object Coding Produced

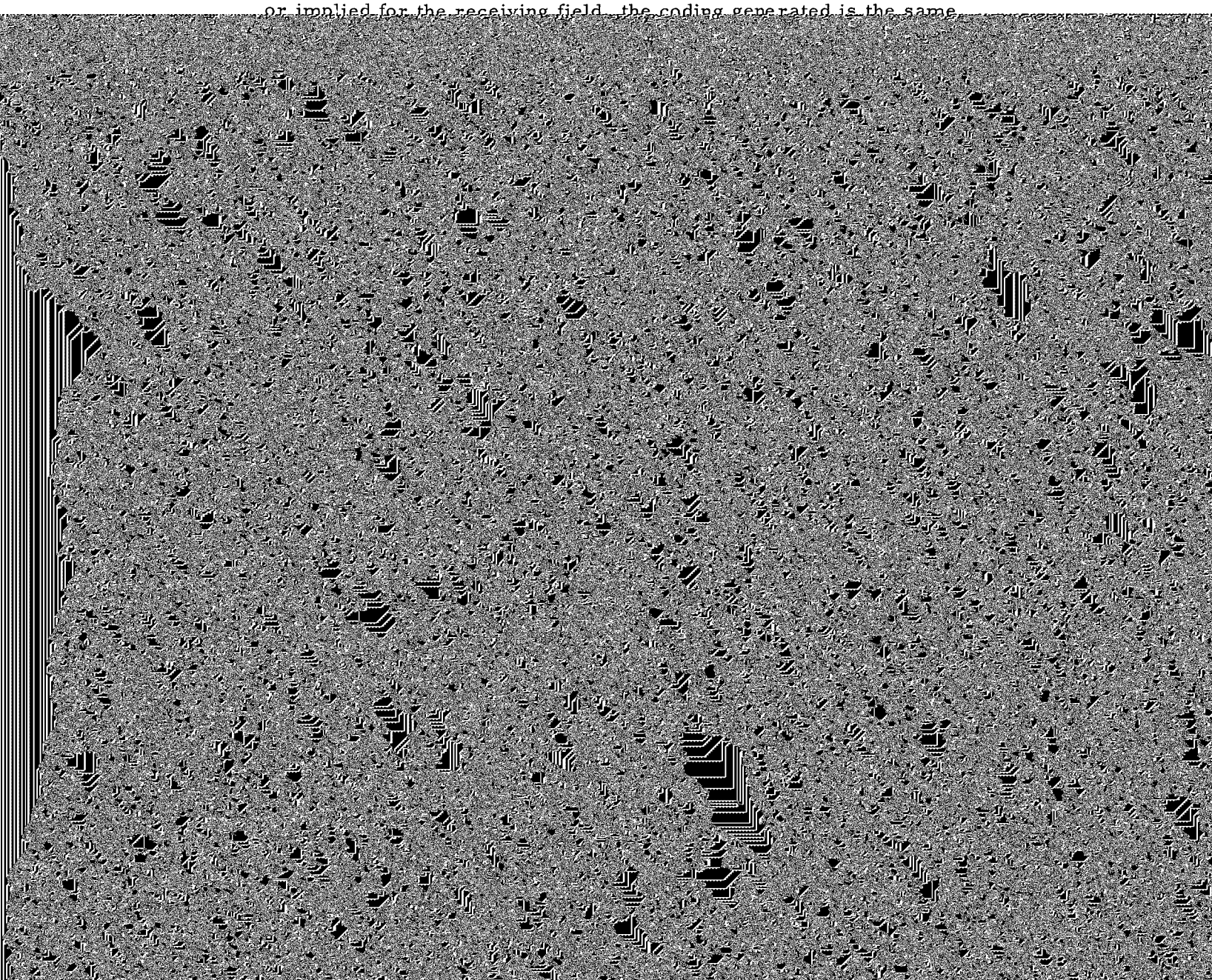
   a. There are four types of MOVE: numeric, non-numeric, ALL, and edited. For a CORRESPONDING [illegible] type of MOVE [illegible]

c. Non-numeric MOVE. For cases in which the sending field is not shorter than the receiving field, from 7 to 16 characters are generated depending upon the preset punctuation in the two fields. When space-fill is required at either end of the receiving field, ALL move coding is generated to fill the entire receiving field with spaces, and then coding is generated as though the two fields were of equal size.

d. ALL Move. Coding is generated which fills the entire receiving field with repetitions of a string of characters. When the receiving field is either elementary or in a redefined area, from 7 to 23 characters of code are generated. The minimum occurs when the receiving field is not redefined and the string consists of a single character. (The figurative constants are considered single character strings.) A redefined receiving field causes the generation of 8 more characters.

When the receiving field is an unredefined group, from 24 to 40 characters are generated depending upon the preset punctuation in the receiving field.

e. Edited Move. When the BLANK WHEN ZERO option is expressed or implied for the receiving field, the coding generated is the same



a SPACE, or a ZERO. A warning diagnostic is produced whenever a character or characters of the sending field must be truncated, either because of decimal point alignment requirements or because the sending field is longer than the receiving field.

c. When the

THRU literal-2

Refers to rightmost character of field only. say: move last 2 chars of field, " 3 chars " "

option is specified, there may be only one receiving field, and literal-2 must be one alphanumeric character.

d. No more than 13 basic operands may appear in a MOVE statement. Each subscripted or edited operand counts as two operands.

e. See Section XI for a complete description of MOVE CORRESPONDING.

MULTIPLY

A.13. MULTIPLY

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

VERB FORMAT:

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

TECHNICAL NOTES:

1. Object Coding Produced

   a. Basically, the coding generated for this verb is a 13-character subroutine calling sequence. To this must be added 8 characters for each operand which is redefined. For those cases where the receiving field is edited, or has insufficient positions to contain the product, or in which SIZE ERROR is specified, a move is generated according to the rules specified under the MOVE verb (Numeric MOVE, Edited MOVE).

   b. If ROUNDED is specified, 7 more characters are generated.

   c. If SIZE ERROR is specified, 16 more characters are generated.

2. General

   a. Each MULTIPLY verb statement must contain at least two operands (i.e., a multiplier and a multiplicand).

   b. Only numeric fields can be used as operands in a MULTIPLY statement. The use of an alphabetic field results in a fatal diagnostic. The use of an alphanumeric field results in a warning diagnostic. The only figurative constant that can be used is ZERO(E)(S).

   c. Of the operands used, the rightmost must be a data-name. Any data-name used can reference an elementary item only. Neither the multiplier nor the multiplicand can contain editing symbols.

   d. Each operand can have an operational sign and an implied decimal point. If no decimal point is 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.

   e. Literals and constant fields cannot be used to receive the product.

   f. Truncation of left and/or right digits of the product can occur during the storage of that product according to the size associated with the receiving field. In general, the rules for truncation are:

      1) When the receiving field contains fewer integer places than the product, left truncation will occur.

      2) When the receiving field contains fewer decimal places than the product, right truncation will occur.

      It should be noted, however, that under certain conditions, right truncation of integer places and left truncation of decimal places is possible. For example, given the product 645821Λ and a

receiving field with a PICTURE of 9999PPPV, decimal point alignment would be,

```
0645 82 1Λ
9999PPPV
```

and the digits stored would be 0645. Similarly, given the product Λ8259723 and a receiving field with a PICTURE of VPPPP9999, decimal point alignment would result in

```
Λ8 2 5 97230
VPPPP9999
```

and the digits stored would be 7230.

g. SIZE ERROR and ROUNDED options are provided for those cases where either or both types of truncation can take place.

1) When the SIZE ERROR option is specified, a test is made to see if left-digit truncation will occur when the product is to be stored in the receiving field. If left truncation will occur, the product is not stored. Instead, the "any imperative statement" associated with the SIZE ERROR option is performed.

2) 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 will occur, the least significant digit of those digits that can be stored is increased by 1 when the most significant digit of those that would be truncated is in the range 5 through 9.

3) If the SIZE ERROR option is not specified and a size error condition occurs, the value of the result is unpredictable.

h. Editing symbols are permitted in the receiving field for the product so long as that field is not used as a multiplier or a multiplicand.

i. Unless the

GIVING data-name-2

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 to:

A · B = B and A · B = C

NOTE

A.14. NOTE

FUNCTION: To permit the programmer to write statements in the PROCEDURE DIVISION which will be produced on the compilation listing but will not be compiled.

VERB FORMAT:

NOTE ... .

TECHNICAL NOTES:

1. Object Coding Produced

   No object coding is produced by the NOTE verb statement.

2. General

   a. When NOTE is the first word inside 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. *

   b. 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.

   c. Any combination of characters from the COBOL set can follow the word NOTE, except in the case of b., above, where the occurrence of a period followed by a space is regarded as ending the "note." i.e. reserved words + anything else can be used.

Programmer

In the case where a whole paragraph is a "note," the paragraph must still follow the proper format.

NOTE statements can occur in the ENVIRONMENT and the DATA DIVISIONS but not the IDENTIFICATION DIVISION of the source program. When inter-compiler compatibility is a concern, however, NOTE statements should be restricted to the PROCEDURE DIVISION. *

e.g. Para 1.
Note .....

Para 2.

In this case the compiler will ignore everything upto the next Para. heading. So you can use periods within the "Note" Paragraph.

OPEN

A.15. OPEN

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

VERB FORMAT:

OPEN [INPUT file-name-1 [WITH NO REWIND][, INPUT file-name-2]]
[OUTPUT file-name-m [WITH NO REWIND][, OUTPUT file-name-m+1]]

TECHNICAL NOTES:

1. Object Coding Produced

   a. An eight-character subroutine calling sequence is generated.

   b. The general action accomplished by the object coding generated for each option of this verb is shown in the following table. The definitions of the symbols in the table are given in the notes which follow it.

		Peripheral file	Peripheral file re-assigned to tape	Single or multiple-reel file	File on a multiple-file tape
OPEN file name	INPUT	E	A, D	A, D	A, F, G, D
	OUTPUT	C	A, B	A, B	A, F, B
OPEN file name NO REWIND	INPUT	E	D	D	G, D
	OUTPUT	C	B	B	H, B
Automatic open of later reel due to CLOSE REEL or automatic tape swap	INPUT		A, D	A, D	
	OUTPUT		A, B	A, B	

Table Notes:

Symbol		Meaning
A	—	The reel is rewound
B	—	If labels are standard, a 1HDR label is created and written on the device. If labels are omitted, a dummy 1HDR label is created and written on the device.
C	—	If labels are standard, a 1HDR label is created and written on the device. If labels are omitted, no action takes place; nothing is written on the device.
D	—	If labels are standard, a block is read and checked as a 1HDR label. If labels are omitted, a block is read and ignored.
E	—	If labels are standard, the device is read and the information checked as a 1HDR label. If labels are omitted, no read or checking takes place.
F	—	The position of this multiple file tape is set to 0.

Symbol		Meaning
G	—	If this file has a position less than or equal to the position of the last file to use this multiple file tape, the tape is searched in a backward direction for the proper 1HDR label. If this file has a position greater than the position of the last file to use this multiple file tape, the tape is searched in a forward direction for the proper 1HDR label.
H	—	The tape is searched in a forward direction for the first 1HDR label or 1ERI label. The tape is then backspaced over this label.

2. General

   a. At least one file must be named when the OPEN verb statement is used.

   b. Both input and output files can be opened in the same OPEN verb statement. The word INPUT or OUTPUT applies to each file following it until another INPUT or OUTPUT is reached or the OPEN statement is concluded.

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

   d. 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.

   e. 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).

   f. When a tape file is ASSIGNed to more than one tape unit, the first reel of that file is OPENed on the first named tape unit, and the following reels are processed alternately from the two tape units.

   g. 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.

   h. If an input file is not found when searching in the indicated direction on a multiple-file tape, the computer stops with an error condition indicated. Starting the computer causes searching to resume in the opposite direction.

PERFORM

A.16. PERFORM

FUNCTION: To depart from the normal sequence of procedures in order to execute one statement, or a sequence of statements, a specified number of times, or until a condition is satisfied, and to provide a means of return to the normal sequence.

VERB FORMAT:

Option 1:

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

Option 2:

PERFORM procedure-name-1 [THRU procedure-name-2] {data-name-1 / integer-1} TIMES

Option 3:

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

Option 4:

PERFORM procedure-name-1 [THRU procedure-name-2]
VARYING data-name-1 FROM {data-name-2 / literal-1} BY {data-name-3 / literal-2} UNTIL condition-1

TECHNICAL NOTES:

1. Object Coding Produced

   Every PERFORM clause is the equivalent of a sequence of other COBOL clauses such as ALTER, GO TO, MOVE, IF, etc., and could be written in the source program as such. The coding generated for PERFORM is the same as if the equivalent clauses had been used by the source programmer.

2. General

   a. When only procedure-name-1 is given:

      1) When procedure-name-1 is a paragraph-name, only that paragraph is executed.

      2) When procedure-name-1 is a section name, all of the paragraphs that make up that section are executed.

   b. When

      procedure-name-1 THRU procedure-name-2

      is given, the coding from the first statement in procedure-name-1 through the last statement of procedure-name-2 is executed. The only necessary relation between procedure-name-1 and procedure-name-2 is that a sequence beginning at procedure-name-1 logically must arrive at procedure-name-2. That is, GO's and PERFORM's can occur between procedure-name-1 and procedure-name-2; however, if there are two or more paths by which procedure-name-2 can be logically reached from procedure-name-1, procedure-name-2 must be a paragraph consisting of the verb EXIT. In other words, when the loop specified by the PERFORM verb state-

ment has more than one possible path to follow, an EXIT paragraph must provide the common ending point for each path.

c. 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.

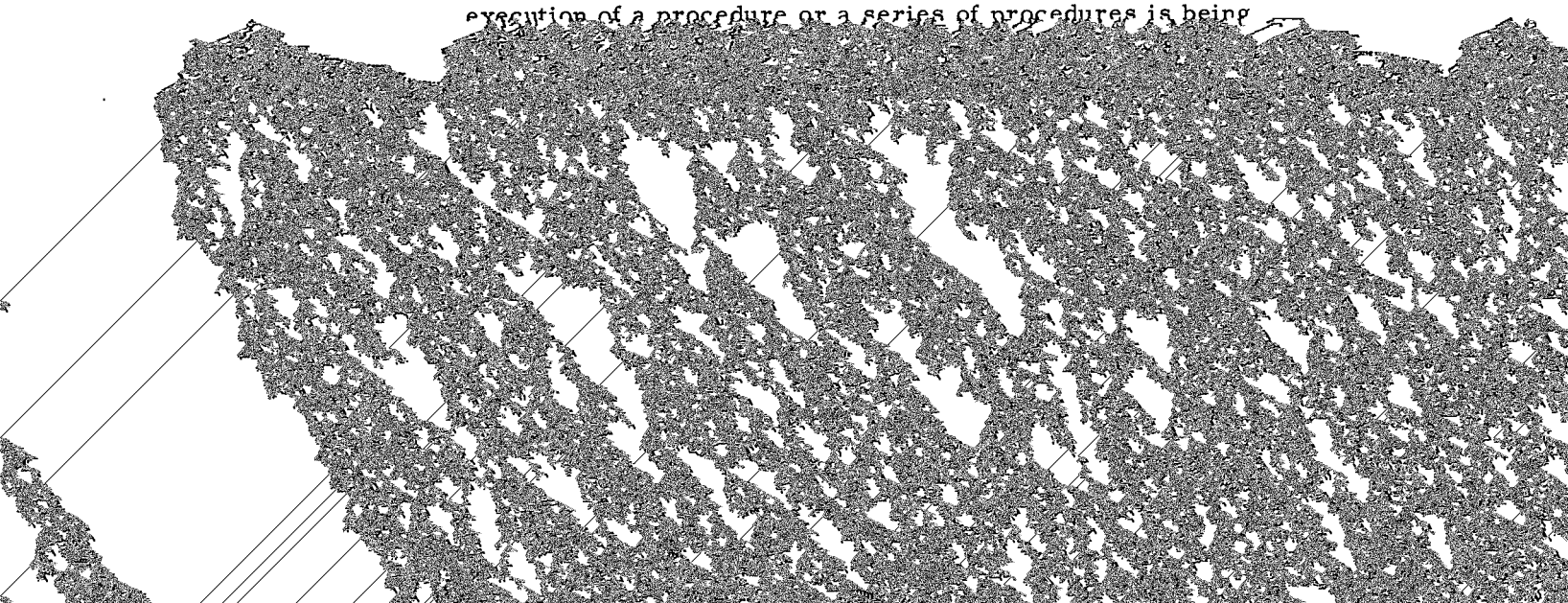
d. The PERFORM mechanism for options 1 through 4 operates as follows, with note c. above applying to all options:

1) Option 1 is a simple PERFORM. A return to the statement following the PERFORM is inserted after the "last statement," and sequence control is sent to procedure-name-1.

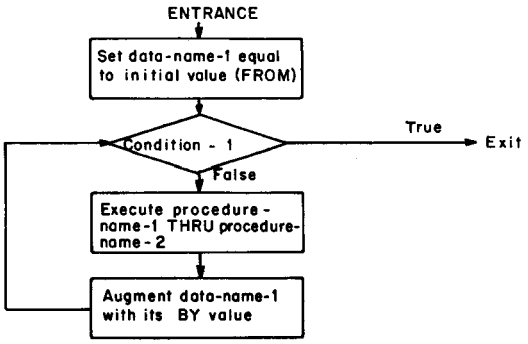
2) Option 2 is the TIMES option. The specified number of times must be a positive integer, and may be zero. The PERFORM mechanism sets up a counter and tests it against the specified value before each jump to procedure-name-1. The return mechanism after the "last statement" returns control to the point at which the counter is incremented, steps the counter, and then sends control to the test. The test cycles control to procedure-name-1 the specified number of times and, after the last time, sends control to the statement following the PERFORM.

3) Option 3 is the UNTIL option. This option is the same as the TIMES option except that an evaluation of a condition takes the place of counting and testing against a specified integer. The condition can be any simple or compound-condition. That is, the condition can involve relations and tests. When the condition is satisfied (i.e., true), control is transferred to the next statement after the PERFORM statement. If the condition is true when the PERFORM is entered, no transfer to procedure-name-1 takes place; control is transferred to the next statement after the PERFORM statement.

4) Option 4 is the VARYING data-name option. The VARYING option is assumed to be arithmetic, and all the arithmetic rules apply. This option should be used when it is desired to increase or decrease the value of any item while the execution of a procedure or a series of procedures is being



until the condition is determined to be true, at which point control is transferred to the next statement after the PERFORM statement. Literal-1 and literal-2 must be numeric literals but need not necessarily be integral. A diagram for this mechanism follows:



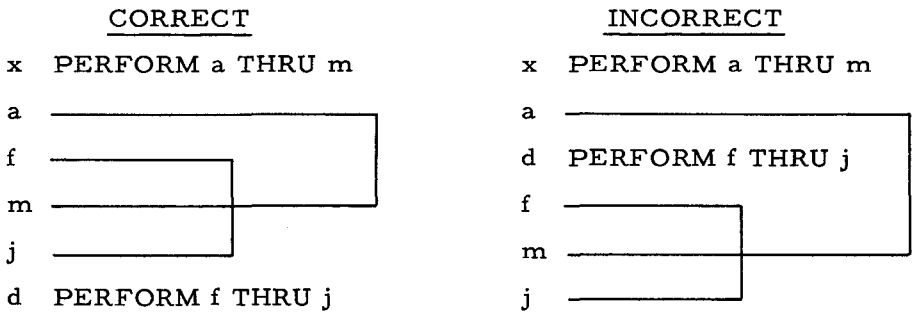
It should be noted that after completion of the PERFORM, data-name-1 will have a value which exceeds its last used value by one increment or decrement, whichever the case may be.

e. In general, procedure-name-1 should not be the next statement after the PERFORM. If it is, the result is that the loop will be executed one more time than was probably intended, because after the PERFORM is satisfied, control would go to procedure-name-1 in the normal continuation of the sequence.

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:



g. A PERFORM which is in a SECTION with a priority number equal to or greater than the SEGMENT LIMIT can have within its range only

1) SECTIONS with the same priority number as that containing the PERFORM, or

2) SECTIONS with a priority number less than the segment limit.

A PERFORM in a section with a priority number less than the SEGMENT LIMIT can have within its range any number of SECTIONs of any priority whatsoever.

A.17. READ

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

VERB FORMATS:

Compiler D Format (best left out.)

READ file-name (RECORD) AT END any imperative statement

Compiler H Format

READ file-name RECORD [INTO data-name] AT END any imperative statement

TECHNICAL NOTES:

1. Object Coding Produced

   When the AT END statement for this verb consists solely of a GO TO clause, 8 characters are generated. In all other cases, 12 characters are produced.

2. General

   a. A READ cannot be executed for a file unless the file is OPENed.

   b. 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.)

STOP

A.18. STOP

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

SUBTRACT

A.19. SUBTRACT

FUNCTION: To subtract one, or the sum of two or more, numeric data item(s) from a specified item and set the value of an item equal to the result.

VERB FORMAT:

Option 1:

SUBTRACT { literal-1 / data-name-1 } [, [{ literal-2 / data-name-2 } ...] FROM [{ literal-m / data-name-m } GIVING]
data-name-n [ROUNDED] [; ON SIZE ERROR any imperative statement]

Option 2:

SUBTRACT CORRESPONDING data-name-1 FROM data-name-2 [ROUNDED] [; ON SIZE ERROR any imperative statement]

TECHNICAL NOTES:

1. Object Coding Produced

   a. The basic coding generated is 7 characters per operand (or 15 characters per redefined operand), excluding the receiving operand when the GIVING option is not used.

   b. A common point location is determined (equal to the largest of the point locations in all the operands). Then, for each operand with a point location not the same as the common point location, coding is produced to move the field to a temporary field. This coding is that generated for a Numeric-MOVE (see MOVE verb) except that instructions whose only function is to normalize the sign are suppressed.

      If the SIZE ERROR or the ROUNDED clause or both are present, or if the receiving field is edited, a move to the receiving field is generated, even when the point location of an operand is the same as the common point location.

   c. When the SIZE ERROR clause is present, 10 more characters are produced; and if the receiving field is defined with fewer integer places than the largest of the integer places in the subtrahends, 12 more characters are generated.

   d. When the ROUNDED clause is present, 7 more characters are produced.

2. General

   a. Each SUBTRACT verb statement must contain at least two operands (i.e., a subtrahend and a minuend).

   b. Only numeric fields and numeric literals can be used as operands in a SUBTRACT verb statement. The use of an alphabetic field results in a fatal diagnostic. The use of an alphanumeric field results in a warning diagnostic. The only figurative constant that can be used is ZERO(E)(S).

   c. Of the operands used, the rightmost must be a data-name. Any data-name used must reference an elementary item only. Neither the minuend nor the subtrahend can contain editing sym-

bols. The rightmost, and only the rightmost, data-name can contain editing symbols. When it does contain such symbols, it must be a receiving field not used in the computation.

d. 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 no decimal point is indicated for a SUBTRACT operand, it is assumed to be logically to the right of the least significant digit of the operand.

e. Literals and constant fields cannot be used to receive the result.

f. When dealing with multiple subtrahends, the effect of the subtraction is as though the subtrahends were first summed, and the sum then subtracted from the minuend.

g. Operands are aligned according to implied decimal points. Zeros are filled in on either end as necessary; that is, 99∧9 and 9∧999 would be aligned and zero filled:

99∧900
09∧999.

h. Truncation of left and/or right digits of the remainder can occur during the storage of that remainder according to the size associated

in the receiving field. If right truncation will occur, the least signficant digit of those digits that can be stored is increased by 1 when the most significant digit of those that will be truncated is in the range 5 through 9.

3) If the SIZE ERROR option is not specified and a size error condition occurs, the value of the result is unpredictable.

j. Editing symbols are permitted in the receiving field for the remainder so long as that field is not used as a subtrahend or a minuend.

k. Unless the

GIVING data-name-n

clause is used, the remainder is stored in data-name-m. When the GIVING option is used, the remainder is stored in data-name-n (which is not used in the arithmetic process). That is,

SUBTRACT A FROM B

is the logical equivalent of

B - A = B

while

SUBTRACT A FROM B GIVING C

is the logical equivalent of

B - A = C.

l. For a complete description of SUBTRACT CORRESPONDING see Section XI.

m. Not more than 13 basic operands may appear in a SUBTRACT statement. Each subscripted or edited operand counts as two operands.

USE

A. 20. USE

FUNCTION: To specify procedures for input/output error and label handling in addition to those procedures supplied by the input/output system.

VERB FORMAT:

Option 1:

USE AFTER STANDARD ERROR PROCEDURE ON file-name-1 [file-name-2 ...]

Option 2:

USE { BEFORE / AFTER } STANDARD [{ BEGINNING / ENDING }] LABEL PROCEDURE ON file-name-1 [file-name-2 ...]

TECHNICAL NOTES:

1. Object Coding Produced

   The basic coding produced for this verb is four characters.

2. General

   a. In the source program, a USE verb statement must only be written following a section header of the DECLARATIVES section. The remainder of the section must consist of one or more procedural paragraphs defining the procedures to be used. A USE DECLARATIVE is used to specify procedures which are to supplement the standard procedures provided by the input/output system. The USE sentence immediately follows the section header and identifies the conditions calling for the execution of the USE procedures. Only PERFORM statements can reference all or part of a USE section. The USE sentence itself is never executed. Within a USE procedure, there must be no reference to the "main body" of the PROCEDURE DIVISION.

      The format for the USE declarative is:

```
DECLARATIVES.
section-name SECTION.  USE ...
paragraph-name.
  first procedure-statement....
```

   b. The procedures designated by valid USE verb statements (see d., below) are executed by the input/output system at the appropriate time in object program execution:

      1) After completing the standard input/output error routine (option 1 USE).

      2) Before a beginning or ending input label check is accomplished (option 2 USE).

      3) After a beginning or an ending output label is created, but before it is placed on the output medium, i.e., magnetic tape or punched card (option 2 USE)

   c. The only valid applications of the USE verb statement are as shown in the following table:

	PRINTER	CARD-FILE				TAPE-FILE			
		IN LABELS		OUT LABELS		IN LABELS		OUT LABELS	
		S	O	S	O	S	O	S	O
USE AFTER STANDARD ERROR PROCEDURE ON....	V	V	V	V	V	V	V	V	V
USE BEFORE STANDARD BEGINNING...	N	V	N	N	N	V	N	N	N
USE BEFORE STANDARD ENDING...	N	V	N	N	N	V	N	N	N
USE AFTER STANDARD BEGINNING...	N	N	N	V	N	N	N	V	N
USE AFTER STANDARD ENDING...	N	N	N	V	N	N	N	V	N

N = Not valid, O = Omitted, S = Standard, and V = Valid

d. When option 2 is specified, if BEGINNING or ENDING is not included, the designated procedures are executed for both beginning and ending labels, where valid.

e. Double USE verb statements cannot be given. That is, only one USE BEFORE STANDARD BEGINNING... statement can be applicable to a file when it is OPEN.

f. USE statements, when present, must be grouped together at the beginning of the PROCEDURE DIVISION. They must be preceded by the word DECLARATIVES and must be immediately followed by the words END DECLARATIVES:

```
PROCEDURE DIVISION.
DECLARATIVES.
section-name-1 SECTION.  USE ...
section-name-2 SECTION.  USE ...
.
.
.
section-name-n SECTION.  USE ...
END DECLARATIVES.
```

END DECLARATIVES must be followed by a section-name.

g. There must be no OPEN, CLOSE, READ, or WRITE verb in a USE DECLARATIVES section.

WRITE

A.21. WRITE

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

VERB FORMATS:

Compiler D Format

```
WRITE record-name [BEFORE ADVANCING { data-name-1 LINES }]
                                    { integer LINES      }
                                    { mnemonic-name      }
```

Compiler H Format

```
WRITE record-name [FROM data-name-1] [{BEFORE} ADVANCING { data-name-2 LINES }]]
                                     {AFTER }            { integer LINES      }
                                                         { mnemonic-name      }
```

TECHNICAL NOTES:

1. Object Coding Produced

   The basic coding produced for this verb is four characters. If the file being WRITten has variable-size records, four more characters are generated. If the file being WRITten is a print file, two more characters are generated. Further, when a data-name appears as an ADVANCING operand, eight more characters are generated, plus eleven more characters if that data-name is redefined.

2. General

   a. A WRITE cannot be executed for a file unless the file is OPEN.

   b. After the execution of a WRITE, record-name is no longer available.

   c. If the FROM option is specified, the data is moved from the area specified by data-name-1 to the output area (following the rules for MOVE without the CORRESPONDING option).

   d. The ADVANCING clause is used only for printer spacing control. In this clause, neither data-name-1 nor integer can specify a line advance greater than 15. The integer, or data-name, is interpreted modulo 16. Mnemonic-name must be the name assigned to one of the following:

      PAGE

      for Type { 222 } Printer or
               { 206 }

      CHANNEL a OF PRINTER m

      on paper-tape loop for Type 222 Printer in the SPECIAL-NAMES paragraph. (See Section IV.)

SECTION VII
OBJECT PROGRAM INPUT/OUTPUT

I/O DATA AREAS

The following pages contain general information concerning the allocation of object memory space to file buffers and current record areas and also indicate the source language statements that control such allocation. The object I/O coding is capable of handling card, punch, printer, and tape files with either single buffering or double buffering. Various combinations of fixed or variable record sizes and blocked or unblocked records may also be accommodated. All of these variables affect the allocation of buffers as indicated.

Buffering

Peripheral Data Transfer.

A buffer is an area of object memory used to hold the data that is read, or written, with one PDT instruction; a buffer may hold one card image, one tape block, or one printer line. A file is allowed one or two buffers, according to the RESERVE NO ALTERNATE AREA clause in the source program. If the RESERVE NO ALTERNATE AREA clause is applied to a file, only one buffer is allocated for that file, and it is said to be single-buffered. All other files receive two buffers and are said to be double-buffered. Reading and writing on single-buffered files can never be overlapped with central processor operations within the same COBOL object program; however, they may be overlapped with other I/O operations. Double-buffered files provide a maximum of overlap with both central processor and I/O operations.

All buffers are followed by a "stopper character" (a single character with a record mark). This character terminates any reading of data that would overflow the input buffer or terminates the writing from an output buffer.

All files in a Series 200 COBOL object program have a current record area. The most recently READ record can be addressed in this area if the file is an input file. If the file is an output file, the next record to be WRITten can be assembled in the current record area. The current record area is in a fixed location for the respective file and is given an abolsute address by the generated object code. The current record area is equal in size to the largest record of the file. It may have word- and item-mark punctuation set on items within it, but no record marks are ever set within such a record area. The current record area for a file may appear within a buffer of that file, or it may be external to the buffers. All external current record areas are followed by an extra character on which record mark punctuation is set. This character acts as a "stopper" when the record is moved by the I/O object code.

Note that the stopper characters associated with external current record areas and buffers are not considered to be part of the record areas or buffers. (This should be kept in mind when estimating the total reserved area for a file.)

Blocking

A block is a unit of data that is either read or written on tape with one PDT instruction. The block may contain one or more logical records plus some control information. Under no circumstances can a partial record appear in a block. Blocking of records is applicable only to files assigned to tape. Peripheral files are always unblocked; that is, each WRITE or READ results in the execution of a PDT instruction. Blocking of tape files is controlled by the BLOCK CONTAINS... clause in each file description. The BLOCK CONTAINS... clause may be omitted, or it may be expressed as some number of RECORDS or some number of CHARACTERS. Omitting the BLOCK CONTAINS... clause for a file in a source program is equivalent to specifying BLOCK CONTAINS 1 RECORD. When blocking of more than one record to a block is desired, the BLOCK CONTAINS... clause is required. Its interpretation, however, depends partially on whether the records of the file are all of the same size or are of different sizes. (This is further discussed below.)

Record Sizes

If the sizes of all records in a file are the same, the file is called "fixed"; if the sizes of the records vary, the file is called "variable."

Peripheral files are always considered to be fixed files. If the actual record descriptions of a print file in a source program yield records of varying size, the largest size is taken as the size of all records. WRITE verbs always cause this maximum size record to be written. Note that on output files (printer and card punch) this restriction may cause shorter records to be "filled" or "padded" with unwanted data from previously written longer records.

The record descriptions of a tape file in a source program may yield varying sizes from a minimum size (MINSIZ) to a maximum size (MAXSIZ). The following combinations of the BLOCK CONTAINS... clause and record sizes should be noted:

CLAUSE	FIXED	VARIABLE
BLOCK CONTAINS 1 RECORD	TYPE A (FORM-1)	TYPE C (FORM-3)
BLOCK CONTAINS n RECORDS*	TYPE B (FORM-2)	TYPE D
BLOCK CONTAINS n CHARACTERS	TYPE E (FORM-4)	TYPE E (FORM-4)
*(where n is greater than 1)		

FORM-1, -2, -3, and -4 refer to files whose RECORDING MODE IS BCD and are terms generally used throughout the industry. For FORM-3 and FORM-4 records, care should be taken to insure

that the data-name given in the SIZE IS ... DEPENDING ON ... clause contains the correct value when the record is WRITten. The I/O code uses the record-name to determine the SIZE and does not check the value in the DEPENDING ON data-name field.

A description of the I/O memory allocation for each type is given below. All peripheral files fall under TYPE A. Types B, C, D, and E pertain to tape files only.

1. Type A Files 1. fixed-length record

   a. Peripheral Type A Files

      1) Card-read and card-punch files have buffers whose size exactly equals the size of the largest record. If the file is single-buffered, one buffer is reserved and the current record area resides within that buffer. If the file is double-buffered, two buffers are reserved and the current record area occupies the second buffer.

      2) A printer file has a buffer whose size exactly equals the size of its largest record. If the file is single-buffered, one buffer is reserved and the current record area resides within that buffer. If the file is double-buffered, two buffers are reserved and the current record area occupies the second buffer. An extra character is reserved to the left of the total buffer area. If the printer file is reassigned to a tape file at object time, this character is incorporated into the adjoining buffer and contains the printer spacing quantity corresponding to the requirements of SCOPE.

      3) Double-buffered peripheral files are handled somewhat differently from double-buffered tape files. The leftmost buffer of double-buffered peripheral files is used to read into or to write from, while the rightmost buffer is used to hold the current record. The I/O object code moves the information from one buffer to the other before executing the next PDT instruction (which always addresses the leftmost buffer). Thus, there is an added transfer of data executed for each record of the file. When double-buffered, peripheral files are reassigned to tape at object time; the technique for writing or reading them remains as described above. It therefore differs from the double-buffering technique used on files originally assigned to tape.

   b. Tape Type A Files

      Type A tape-file buffers contain one record plus the banner character.

area occupies the leftmost record position in the buffer if the file is an input file, and the rightmost record position in the buffer if the file is an output file. Double-buffered files have two buffers with an external record area.

3. Type C Files

Type C tape-file buffers are equal in size to the MAXSIZ plus four characters (one for banner and three for block size). Output buffers have an extra three characters for a trailing block count if the TRAILING-COUNT technique was APPLied to the file. The TRAILING-COUNT technique is ignored on input files. Single-buffered files have one buffer, and the current record begins at the fifth character from the left end of the buffer. Double-buffered files have two buffers and an external, current record area equal to MAXSIZ. Files with RECORDING MODE IS BCD do not use the leftmost-four characters of the buffer.

4. Type D Files

Type D tape-file buffers are equal in size to n(MAXSIZ+2)+4 characters. This allows for a banner character, a three-character block size, and a two-character record size on each record. Output buffers have an extra three characters for a trailing block count if the TRAILING-COUNT technique was APPLied to the file. The TRAILING-COUNT technique is ignored on input files. Single-buffered files have one buffer, with an external current record area equal to MAXSIZ. Double-buffered files have two buffers, with an external current record area equal to MAXSIZ. Double-buffered files have two buffers, with an external current record area equal to MAXSIZ.

5. Type E Files

Type E tape-file buffers are equal in size to n characters. Of these, one is used for a banner character, three are used for a block count, and three are used for a trailing block count if TRAILING-COUNT has been APPLied to the file. The remainder are used to hold data records with a two-character record size for each record. Single-buffered files contain one buffer and an external, current record area equal to MAXSIZ. Double-buffered files have two buffers and an external, current record area equal to MAXSIZ. When RECORDING MODE IS BCD, the leftmost-four characters are used for a block count consisting of four decimal characters.

LABEL RECORDS

Every file description in the source program must contain the phrase: LABEL RECORDS ARE $\left\{\begin{matrix}\text{OMITTED} \\ \text{STANDARD}\end{matrix}\right\}$. The description of a STANDARD label is given below. Label-checking and label-writing routines supplied in the object coding check or create certain fields within a label record. The fields that are not checked or created by the object I/O are the province of the source programmer. They may be accessed only by the names given in the field descriptions and only within USE procedures. The fields in the STANDARD label that are checked on input and created on output are listed below with a description of the label activity that occurs in the object code during OPEN's and CLOSE's.

Description of Standard 80-Character Label

the asterisk is part of the name!

```
01 *STANDARD-LABEL
   02 *SENTINEL PICTURE X(5).
   02 *TAPE-NO.
   03 *BLOCK-COUNT PICTURE X(5).
   02 *RECORD-COUNT.
   03 *FILE-NO. PICTURE X(6).
   03 *REEL-NUMBER PICTURE X(3).
   03  FILLER PICTURE X.
   02 *IDENTIFICATION PICTURE X(10).
   02 *DATE-WRITTEN PICTURE X(5).
   02 *PURGE-DATE PICTURE X(5).
   02 *LABEL-EXCESS.
   03 *SCOPE-PARAMS-1 PICTURE X(4).
   03 *EXCESS-FILLER PICTURE X(33).
   03 *SCOPE-PARAMS-2 PICTURE X(3).
```

Description of Standard 120-Character Label

```
01 *STANDARD-LABEL-120
   02 *SENTINEL PICTURE X(5).
   02 *PURGE-DATE-120 PICTURE X(5).
   02 *DATE-WRITTEN-120 PICTURE X(5).
   02 *IDENTIFICATION-120 PICTURE X(10).
   02 *FILLERA-120 PICTURE X(11).
   02 *REEL-NUMBER-120 PICTURE X(4).
   02 *FILLERB-120 PICTURE X(25).
   02 *CHECK-POINT PICTURE X.
   02 *BLOCK-COUNT-120 PICTURE X(6).
   02 *FILLERC-120 PICTURE X(48).
```

If the preceding names are referenced, the * must be included as part of the name.

Description of Label-Processing

1. OPEN, tape input files, labels STANDARD.
   a. See Object Messages and Error Halts below, for tape positioning at OPEN time.
   b. Read one block into label area.
   c. Check SENTINEL for presence of "1HDRΔ."
   d. Execute USE procedure if present.
   e. Check IDENTIFICATION.

f. Check DATE-WRITTEN, if present in FD.

g. Check REEL-NUMBER against internal reel number, unless this file is on a MULTIPLE-FILE tape.

h. If labels are STANDARD-120, and the *CHECK-POINT field contains a 1, the checkpoint will be bypassed.

i. If a tape mark follows the label, the tape mark will be bypassed.

2. OPEN, tape input files, labels OMITTED.

a. If NO REWIND has not been specified, rewind tape.

b. If RECORDING MODE IS BCD, exit.

c. Read one block into label area.

d. If NO REWIND, backspace.

3. READ, tape input files, labels STANDARD or OMITTED, H-200 tapes for each physical block.

Banner char = J. On all output records & input records. Used to qualify data as it's read by or created by a COBOL program.

a. If first character of block has BA bits equal to "10," release as data; otherwise:

b. If labels STANDARD and USE procedure is present, move 80 characters of block to label area, execute USE.

c. If fourth character of SENTINEL is "R", execute CLOSE REEL.

d. If fourth character of SENTINEL is not "R", execute AT END exit from READ.

occurs as 1st char - 80 char record in ≤ 1 chars was

If you make up your own tapes then J must be at beginning of each block. But if all tapes strictly COBOL created, it takes care of itself.

4. READ, tape input files for each logical record of type B and D files.

a. First character of the record is checked for 77_8, and if present, logical READ is repeated. If RECORDING MODE IS BCD, padding character is 9_{10}; if RECORDING MODE IS H-200, padding character is 77_8.

b. If padding character is not present, record is released normally.

5. READ, tape input files, BCD tapes, for each physical block.

a. If tape mark not detected, release block as data.

b. If tape mark is detected and LABEL RECORDS OMITTED, take AT END path.

c. If tape mark is detected and LABEL RECORDS STANDARD, read next block into label area, execute USE.

d. If first 4 characters of label area are "1 EOR," execute automatic CLOSE REEL; otherwise take AT END path.

6. CLOSE [REEL], tape input file, labels STANDARD or OMITTED.

a. If NO REWIND, exit; otherwise:

b. If LOCK, rewind and release tape, exit; otherwise:

c. If LOCK not given, rewind tape.

d. If REEL given, OPEN next reel.

7. OPEN tape output files, labels STANDARD.

a. See Object Messages and Error Halts below, for tape positioning at OPEN time.

b. Read one block into label area.

c. Backspace one block.

d. Put "1HDRΔ" into SENTINEL.

e. Put reel number into REEL-NUMBER.

f. Put value of IDENTIFICATION into IDENTIFICATION.

g. Put monitor date into DATE-WRITTEN.

h. If peripheral file, put values in SCOPE-PARAMS-1 and SCOPE-PARAMS-2

i. Execute USE procedure if present.

j. If SENTINEL is not "1HDRΔ," halt (see VII-15).

k. Write label on tape.

l. If RECORDING MODE IS BCD, write a tape mark.

8. OPEN tape output files, labels OMITTED.

a. If NO REWIND has not been specified, rewind tape.

b. Read one block into label area.

c. Backspace one block.

d. If NO REWIND was specified or if RECORDING MODE IS BCD, erase tape, exit; otherwise:

e. Put "1HDRΔ" into SENTINEL.

f. If peripheral file, put values in SCOPE-PARAMS-1 and SCOPE-PARAMS-2.

g. Write label on tape.

9. CLOSE [REEL] tape output files, labels STANDARD.

a. Pad any partial buffers of type B and type D files.

If RECORDING MODE IS BCD, padding character is 9_{10}.

If RECORDING MODE IS H-200, padding character is 77_8.

b. Write any unwritten buffer.

c. Blank out label area.

d. Put "1EOFΔ" in SENTINEL, "1EORΔ" in SENTINEL if REEL was specified.

e. Put block count in BLOCK-COUNT.

f. Put value of IDENTIFICATION into IDENTIFICATION.

g. Execute USE procedure, if present.

h. If RECORDING MODE IS BCD, write tape mark.

i. Write label on tape.

j. If RECORDING MODE IS BCD, write tape mark.

k. Put "1ERIΔ" into SENTINEL.

l. If peripheral file, put values in SCOPE-PARAMS-1 and SCOPE-PARAMS-2. Put erase count into characters 53 and 54 of label.

m. Write label on tape twice.

n. Backspace tape twice.

o. If NO REWIND, exit; otherwise:

p. If LOCK, rewind and release tape, exit; otherwise:

q. If LOCK not given, rewind tape.

r. If REEL was specified OPEN next reel.

10. CLOSE [REEL] tape output files, labels OMITTED.

a. Pad any partial buffers of type B and type D files.

If RECORDING MODE IS BCD, padding character is 9_{10}.

If RECORDING MODE IS H-200, padding character is 77_8.

b. Write any unwritten buffer.

c. Blank out label area.

d. Put "1ERIΔ" into SENTINEL.

e. If peripheral file, put values into SCOPE-PARAMS-1 and SCOPE-PARAMS-2. Put erase count into characters 53 and 54 of label.

f. If RECORDING MODE IS BCD, write tape mark.

g. Write label on tape twice.

h. Backspace tape twice.

i. If NO REWIND, exit; otherwise:

j. If LOCK, rewind and release tape; otherwise:

k. If LOCK not given, rewind tape.

l. If REEL given, OPEN next reel.

11. Automatic tape swap on tape output files when end of tape is encountered.

a. Set indicators to perform a CLOSE REEL on file.

b. If labels STANDARD, execute 9.c. through 9.r.

c. If labels OMITTED, execute 10.c. through 10.l.

12. OPEN card input file, labels STANDARD.

a. Read one card into label area.

b. Check SENTINEL for "1HDRΔ."

c. Execute USE procedure if present.

d. Check IDENTIFICATION.

e. Check DATE-WRITTEN, if present in FD.

13. OPEN card input file, labels OMITTED.
No card movement.

14. READ card input file, labels STANDARD.

a. Check first five characters of each card for "1EOFΔ," and take AT END exit if found; otherwise:

b. Release card as data.

15. READ card input file, labels OMITTED.
    Release all cards as data.

16. CLOSE [REEL] card input file, labels STANDARD or OMITTED.
    a. If REEL given, CLOSE is ignored; otherwise:
    b. No card movement.

17. OPEN punch file, labels STANDARD.
    a. Blank label area.
    b. Put "1HDRΔ" into SENTINEL.
    c. Put value of IDENTIFICATION into IDENTIFICATION.
    d. Put monitor date into DATE-WRITTEN.
    e. Execute USE procedure if present.
    f. Punch label card.

18. OPEN punch file, labels OMITTED.
    No card movement.

19. CLOSE [REEL] punch file, labels STANDARD.
    a. If REEL has been given, ignore the CLOSE entirely.
    b. Blank label area.
    c. Put "1EOFΔ" into SENTINEL.
    d. Put value of IDENTIFICATION into IDENTIFICATION.
    e. Put card count into BLOCK-COUNT.
    f. Execute USE procedure if present.

Format of the OECD card

Columns	Contents	Meaning or Action
1-6	ΔΔΔΔΔE	OECD card identification.
7-20	Blank	

If the device is the console, each message (informational or true DISPLAY) is followed by a carriage return. If the device is the printer, each message is followed by a space of four lines. Either the device and/or the spacing of the printer may be altered by placing the proper values in columns 24-26. The values are as follows:

Columns 24 25 26	Meaning
-	printer is to be used for DISPLAY.
2	console is to be used for DISPLAY.
n	new control unit address.
x	any legal printer advancing character.

The following combinations are legal: -ΔΔ, -Δx, -nΔ, -nx, and 2nx. If column 24 is other than - or 2, all three columns are ignored.

3. Columns 27-80

These columns allow changes to be made to a maximum of nine files of the object program. Six columns are required for each file change. The six columns are called a file media field. One of the six columns is used to identify the file number. The files can be found numbered, from 1 to 9 and B to G, on the source listing. Change requests may

If column 1 does not contain a "1" and column 4 does not contain a legitimate file number (1-9, B-G), the entry is ignored. If only one tape was assigned originally, a second tape may be assigned with the OECD card by entries in columns 5 and 6, or these columns may be left blank.

b. Peripheral Files

The user may wish to alter certain characteristics of the card reader, or card punch, from the standard set supplied by the COBOL compiler. The compiler supplies coding (for all files assigned to the card equipment) which reads or writes cards in the Hollerith standard mode. In addition, the reader and punch are initialized during the OPEN of each file to reject cards with illegal punches and/or hole-count errors, and to go "busy" when such errors appear. This requires that the operator respond to such errors and clear the error at the card reader. Changes may be made to these conditions for each card or punch file. In addition, any peripheral file may be reassigned to one or two tape units. When changes are being made to a peripheral file in order to alter the conditions under which it is read or written or to reassign it to a new control unit of the same type, the format of the file media field is as follows:

Column No.	Meaning
1	- control unit is printer.
	J control unit is card-reader.
	K control unit is card-punch.
2	control unit address, if it is being changed (0-7) or (-, J, K. . . P).
3	S alter mode to Hollerith SPECIAL.
	T alter mode to transcription.
4	file number from source listing.
5	Hole count control.
	R reject cards with hole-count errors, do not "go busy."
	B do not reject cards with hole-count errors, "go busy."
	I ignore hole-count errors.
6	Illegal punch control.
	R reject cards with illegal punches, do not "go busy."
	B do not reject cards with illegal punches, "go busy."
	I ignore illegal punches.

When changes are being made to a peripheral file to change its control unit address to that of a tape file, the format of the file media field is identical to that of a file originally assigned to tape (see 1., above).

A peripheral file may be reassigned, by use of the OECD card, only to a control unit of the same type or to tape. Thus, an attempt to assign a printer file to the punch is ignored.

Outline of Operation

An OECD card is not required for the successful running of a COBOL object program, provided that the user does not wish to alter any of the object-time assignments indicated by the source program. However, if one of these assignments is to be altered, an OECD card must be punched and placed in the card reader before loading the object program. SENSE Switch 1 must be set to OFF, and the program loaded in the normal way. If SENSE Switch 1 is ON, no OECD card is read. A special segment of the object program checks the SENSE switch setting, reads the OECD card if SENSE Switch 1 is OFF, and makes the necessary alterations in the object program. This segment of code is then overlaid by the remainder of the object code and does not diminish the memory space normally available to the object code.

OBJECT MESSAGES AND ERROR HALTS

There are two types of messages which the object program may convey to the user during object running. One type of message is merely informational in nature. It signals the operator that some milestone has been reached in the object run. There is no need to halt after such a message. The other type of message results from an error condition. It necessitates a halt and probably some action from the operator. These two types of messages are treated quite differently at object time.

Informational messages are displayed at object time on the display device. If the display device is a printer and there is a file open on this printer at object time, the display messages are suppressed. However, as soon as the printer file is CLOSEd (and as long as no print file is OPEN on the display printer), the informational messages appear on it. If the display device is a console typewriter, the informational messages always appear on it, regardless of any print files in the object program. The informational messages are as follows: (Note: In all cases "n" is the control unit address as given in the ASSIGN... clause of the ENVIRONMENT DIVISION of the source program. In the case of tape files, "nd" is the control unit address and tape number ASSIGNed. The file number as assigned by the compiler is shown as "f," and a count of the number of tape blocks or cards that were read or written, including label records, occurs as "xxxxx," a decimal number. For tape files, the reel number in decimal is shown as "rrr.")

END ΔPRT-nΔ FILEΔ f signifies the closing of a print file.

ENDΔ CRD-nΔ FILEΔ fΔCOUNTΔ XXXXX signifies the closing of a card input file.

ENDΔ PCH-nΔ FILEΔ fΔCOUNTΔ XXXXX signifies the closing of a punch card file.

ENDΔ FILE Δ TAPE-ndΔ FILEΔ fΔ REELΔ rrrΔCOUNTΔXXXXX signifies the closing of a tape file.

ENDΔ REEL ΔTAPE-ndΔ FILEΔ f ΔREEL Δrrr Δ COUNTΔXXXXX signifies the closing of an intermediate reel of a tape file. The operator should react to this by removing the reel and mounting a new reel on the same unit.

Error messages always require a halt. Since standards exist for indicating the message in a coded form in the Ac and Bc registers and since the display device is not always available, no attempt is made to print a message on the display device; all information is coded into the Ac and Bc registers.

In general, the operator may react to an error situation in more than one way.

1. He always has the option of discontinuing the object run.
2. In some instances, he may correct the error condition and push the RUN button to continue.
3. In some instances he may exercise a choice of actions by either setting or resetting sense switch one or may even make a third choice by setting the contents of Ac into the sequence register. When the operator indicates his choice via sense switch one, he must execute a certain sequence of actions. If properly done, such a sequence guarantees that sense switch one can still be used by the program as a switch-name. The sequence of actions is as follows:
   a. A halt occurs.
   b. The operator displays the contents of Ac and Bc to determine the type of halt and his course of action.
   c. The operator notes the current setting of sense switch one and then sets the switch to indicate his course of action.
   d. He pushes the RUN button. The program records the switch setting and comes to an immediate halt. (The Ac and Bc registers are meaningless.)
   e. The operator restores sense switch one to its former state and pushes the RUN button again. The object error code now acts on the stored version of the sense switch. (At no time during the running of the object program is the actual sense switch tested by an error routine except immediately after a halt of the type that allows the operator a choice of actions.)

The following table shows all halts that occur in a COBOL object program and the

Ac	Bc	Condition	Action by Operator	SSL Status	Action by Error Routine

Ac	Bc	Condition	Action by Operator	SSI Status	Action by Error Routine

processing continues as though the write error had not occurred. If the error persists after 64 attempts to rewrite the block and no error USE procedures exist, the error halt occurs.

In either case (i.e., whether or not the write was correctable), if error USE procedures are present, they are executed. The field *ERROR-COUNT is available to the USE procedure as in the case of a tape read.

Eliminating Halts at Object Time

When the HLT-CTL option (see Section IV) is specified in the OBJECT-COMPUTER paragraph, all object time halts are deactivated and replaced by DISPLAY/ACCEPT loops. A DISPLAY/ACCEPT loop is a sequence of coding that (1) causes a message to be displayed on the display device and (2) requests a command (one character) to the accept device. The display and accept device must not be the device used for the foreground program. The computer loops on a busy test until the operator supplies a one-character command.

Upon receipt of the command, the program resumes its processing; there is no halt, the STOP light remains off, and the program may be interrupted at any point in the process.

The user may execute ACCEPT and DISPLAY verbs and is urged to replace all "STOP literal" statements with DISPLAY-ACCEPT statements. The presence of HLT-CTL does not alter the "STOP literal" object code. Therefore, if halts are to be avoided, all "STOP literal" statements must be eliminated from the source-language program.

The "STOP RUN" statement (WITH SUPERVISOR CONTROL) processing remains the same; that is, the object run tape is positioned behind the last segment of the object program and Tape Loader-Monitor C is entered at location 202_8 (130_{10}).

Under normal processing, SENSE switch 1 is used to indicate a decision when a halt occurs. If the halt is replaced by a DISPLAY-ACCEPT loop, a character must be entered to indicate the setting of SENSE switch 1. If ON is the desired setting, a "1" should be entered; if OFF is the desired setting, a "0" should be entered.

When a halt occurs, the operator normally pushes the RUN button to process the next sequential instruction. When HLT-CTL has been specified, the operator must enter a "G" to emerge from the DISPLAY-ACCEPT loop.

Thus, when a DISPLAY-ACCEPT loop is initiated, the operator must take the corrective action indicated in the table of object program halts (see above). The operator must indicate the setting

of SENSE switch 1 (0 or 1) and then direct the program to continue (G). If SENSE switch 1 is not used for a halt condition, the operator may simply enter "G" after correcting the condition.

The following messages may appear when HLT-CTL has been specified:

B Address	Typewriter Message
0C0X	"UNITΔΔCCΔXΔNOTΔREADY □
0C1X	"TPΔERΔCCΔX □
or	or
0C2X	"IOΔERΔCC □
0C3X	"ENDΔTAPEΔONΔMFT □
	"FILEΔXΔNOTΔFOUNDΔONΔMFT □
0C4X	"TAPEΔΔCCΔXΔCHANGE □
0C7X	"FILEΔXΔLBLERRΔaΔ □
	(where a is a low order character in AC register of halt.)
42XX	"FILEΔFOUNDΔLOCKED □
41XX	"FILEΔXΔUSEDΔBEFOREΔOPEN □
43XX	"FILEΔXΔNOTΔOPENΔATΔCLOSE □
44XX	"FILEΔXΔOPENEDΔTWICE □
45XX	"FILEΔXΔREADΔAFTERΔENDΔEXIT □
46XX	"FILEΔXΔPRESENT? □
0500	"FINALΔPARAGRAPHΔEXITΔUNDEFINED □

TAPE POSITIONING AT OPEN TIME

When a tape file is OPENed at object time, two source-program phrases affect the tape positioning that takes place. These are the OPEN statement itself:

OPEN $\left\{\begin{matrix}\text{INPUT}\\\text{OUTPUT}\end{matrix}\right\}$ file-name WITH NO REWIND

and the MULTIPLE-FILE clause in the ENVIRONMENT DIVISION: MULTIPLE-FILE TAPE CONTAINS file-name-1, file-name-2, file-name-3, etc.

When the MULTIPLE FILE clause is used, the following rules must be observed:

1. All files on the multiple file tape (MFT) need not be described and named in the statement, but those that are described must be named in the order in which they appear on the tape. i.e. Name the files which you are going to use.
2. All files on the MFT, whether described or not, must have standard 80 character label records.
3. When a physical file on an MFT is both written and read in the same program, its input file-name must appear immediately ahead of its output file-name in the MFT statement. Thus, if files A, B, and C are written on an MFT in that order and later read back under the names D, E, and F, respectively, the MFT statement would read:

   MULTIPLE FILE TAPE CONTAINS D, A, E, B, F, C.
4. No more than four MULTIPLE FILE statements may be given in one source program.

5. All files that are named in any one MULTIPLE FILE statement must be assigned to the same device and, if reassigned at object time, must all be reassigned to the same device.

6. When a file is OPENed and READ from an MFT, the file may be CLOSEd before the AT END exit of the READ is taken. If the file is CLOSEd WITH NO REWIND, the tape is left positioned in the middle of the data.

7. No file whose RECORDING MODE IS BCD may be named in an MFT statement.

Positioning of Files on Multiple File Tapes (MFT)

For each MULTIPLE FILE statement in the source program, one character is reserved in the object core to contain the position of the most recently used file on that tape. This character is called the MFT character in the discussion below.

1. OPEN OUTPUT

   a. WITH NO REWIND present.

      The tape is searched in a forward direction for the first begin-file label or the end-of-information record and backspaced over this label. The file's position is entered in the MFT character.

   b. WITH NO REWIND not present.

      The tape is rewound. The file's position is entered in the MFT character.

2. OPEN INPUT

   a. WITH NO REWIND present.

      The file's position is compared with the MFT character. If the position is less than the MFT character, the tape is searched backward for the desired file header label. If the position is greater than or equal to the MFT character, the tape is searched forward for the desired header label. The file's position is then entered in the MFT character. If the beginning of tape or the end of information label is encountered during the search, the search direction is reversed and an error halt occurs. Upon starting, searching resumes in the opposite direction. This process is repeated as many times as the operator depresses the START button.

   b. WITH NO REWIND not present.

      The tape is rewound, and the MFT character is set to 0. The tape is searched forward for the desired file header label. The file's position is then entered in the MFT character.

3. CLOSE all files

   a. WITH NO REWIND present.

      No tape movement takes place. The file's position remains in the MFT character.

   b. WITH NO REWIND not present.

      The tape is rewound, and the MFT character is set to 0.

Positioning of Files on Non-Multiple File Tapes

Note that files that are reassigned at object time from peripheral devices to tape are included in this category.

1. OPEN OUTPUT
   a. WITH NO REWIND present.
      No tape movement takes place.
   b. WITH NO REWIND not present.
      The tape is rewound.
2. OPEN INPUT
   a. WITH NO REWIND present.
      No tape movement takes place. No searching is done.
   b. WITH NO REWIND not present.
      The tape is rewound. No searching is done.
3. CLOSE all files
   a. WITH NO REWIND present.
      No tape movement takes place.
   b. WITH NO REWIND not present.
      The tape is rewound.

Peripheral Files

TAPE FILE FORMATS

COBOL object programs are capable of handling tape files that conform to either of two conventions. These conventions correspond to Honeywell bannered-file formats and IBM BCD (Binary-Coded Decimal) file formats.

Honeywell bannered files (recorded in odd parity on half-inch magnetic tape) are fully documented earlier in this section. COBOL object programs are capable of reading and writing any of these formats and provide full compatibility with tapes written by Honeywell input/output

label when opening a reel of a file and always writes a tape mark after this label block. However, when reading such a tape, the COBOL object program does not require the presence of this tape mark and ignores it if it is present. Thus, an input reel of tape that contains no data blocks must have a dummy data record or two tape marks between the header and trailer records.

2. When labels are described as STANDARD-120, the COBOL object program allows a checkpoint dump to be generated (in either even or odd parity) after the header label. The tape mark between the header label and dump is optional. The checkpoint dump must be followed by a tape mark. When the COBOL object program detects a "1" in the *CHECKPOINT field of the header label, the optional tape mark is passed over and the checkpoint data is bypassed until the final tape mark is found. Note that a *CHECKPOINT field that is improperly set will cause the entire file to be bypassed. The COBOL object program does not write checkpoint dumps on output BCD files but does automatically set the *CHECKPOINT field to "0" in the standard 120-character label that it creates.

3. When the COBOL object program is READing a BCD tape and a tape mark is encountered, the end of data is assumed. If the labels have been described as standard (80 or 120 characters), the next block is read into the label area. Noise records are bypassed and error correction attempted, but a non-correctable READ does not cause a halt. If the first five characters of the block are not 1EORΔ, the file is assumed to be AT END. If the block does contain 1EORΔ, an automatic reel swap occurs. If the labels have been described as OMITTED, no READ takes place and the file is assumed to be AT END. Note that files of the following configurations may be described with standard labels and be accepted as input to the COBOL object program:

   a. 1HDR label, data, TM (tape mark)

   b. 1HDR label, TM, data, TM

   c. 1HDR label, TM, data, TM, $\left\{\begin{matrix}\text{1EOF}\\\text{1EOR}\end{matrix}\right\}$ label

   d. 120-character 1HDR label, checkpoint, TM, data, TM

   e. 120-character 1HDR label, TM, checkpoint, TM, data, TM

   f. 120-character 1HDR label, checkpoint, TM, data, TM

      $\left\{\begin{matrix}\text{1EOF}\\\text{1EOR}\end{matrix}\right\}$ label

   g. 120-character 1HDR label, TM, checkpoint, TM, data, TM,

      $\left\{\begin{matrix}\text{1EOF}\\\text{1EOR}\end{matrix}\right\}$ label

   In addition, the following file configurations may be described with label OMITTED and be accepted as input to the COBOL object program:

   a. data, TM

   b. data, TM, 1EOF label

Non-Standard Formats

IBM BCD tapes can be read or written by COBOL object programs only on a Series 200 computer with the 051 code conversion option. When data from a BCD tape is read into memory,

it is identical to data read from a Honeywell odd-parity bannered tape file. The full range of COBOL procedure division verbs may be used to manipulate this data. However, the COBOL object program distinguishes the data described in BCD files as DISPLAY-2 usage unless it is explicitly described as other than DISPLAY-2. This distinction allows the COBOL object program to collate this data in the same sequence that IBM calls COMMERCIAL-COLLATE. (See V-45 and VI-22 through VI-26.)

SECTION VIII

COMPILER DESCRIPTION AND OPERATING INSTRUCTIONS

GENERAL

The minimum hardware requirements for Series 200 COBOL Compilers D and H are listed in the Introduction (see Tables 0-1 and 0-2). Both the compiler and the object programs resulting from compilation run under control of Tape Loader-Monitor C. It is assumed that the reader is familiar with Tape Loader-Monitor C operation.

The Series 200 COBOL Compilers can operate under the control of Drum Monitor C. The minimum machine requirements are

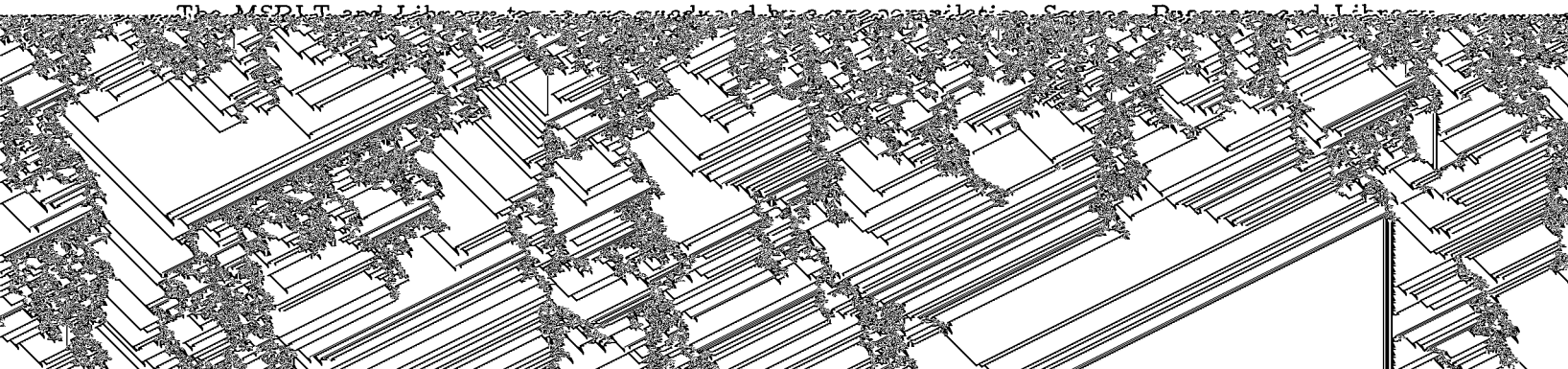
16K characters of core memory
3 magnetic tape units
1 Type 270 Random Access Drum
1 card reader
1 card punch
1 printer.

The Series 200 COBOL Compilers can process source-language programs either singly or in batches. Batched programs can be on cards or on a Master Source Language Program Tape (MSPLT) or on both. The output of compilation is an Object Binary Run Tape (OBRT). The OBRT contains the Tape Loader-Monitor C, Series 200 COBOL specific systems routines, and the compiled programs. Note that any program in a batch that does not compile is not placed on the OBRT. The compiled programs can be called for execution by either visibility or by name.

The order in which compiler output is produced at load-time is:

1. WORKING-STORAGE SECTION,
2. FILE SECTION
3. CONSTANT SECTION, and
4. PROCEDURE DIVISION.

In addition to the input (card decks and/or MSPLT) mentioned above, a Library tape is necessary if any source program to be compiled contains a COPY FROM LIBRARY statement. The MSPLT and Library tape are produced by [illegible] Source Programs and Library



SIX-TAPE SYSTEM

Five-Tape Environment

COBOL Compiler D is a six-tape system, allowing compilations to be run in a complete off-line environment. As part of the six-tape system there is an option for five tapes in which the fifth tape could be used as either an off-line print tape or an input card-image tape.

If the fifth tape is to be used in this capacity, the following rules must be followed:

1. A 101 must be punched in columns 48 through 50 of the ECD card.
2. An S must be punched in column 71 of the ECD card to designate this tape as being a print tape.
3. The tape must be assigned to logical tape unit 1.
4. A printer and/or a console typewriter must be present in this environment to provide a means of communicating with the operator.

The off-line print-image tape system is compatible with Data Conversion A and C except that Data Conversion A and C does not handle multi-reel files. If, however, the end-of-tape is sensed during compilation, the operator is given instructions via the display device to mount a new tape and put it in the "permit" mode. Compilation will then continue using the new tape. However, when the tape-to-print operation is taking place, Data Conversion A and C terminates with the end of one reel. It will then have to be restarted to print the second tape. The system can be used for batched-compilations and/or programs using the library and/or phase tapes,

Parameter	Value	Description
12	Δ	Not used
13	S, T, Δ	User specified
14	02 or Δ	User specified; depends on parameter 13
15	F, S, I	User specified
16	2, 3	User specified
17		User specified
18		User specified
19		User specified
20	02	Printer PCU number
21	40	Tape PCU number
22	1	Depends on parameter 15
23	ABC	Any three alphabetic characters are permissible but three must be present.
24		User specified
25	136	Buffer length needed for conversion
26	3, 4, Δ	User specified
27	C, Δ	Necessary only when the print tape contains more than one type of file.
28	pp, Δ	Dependent on parameter 27
29	Δ	Honeywell format
30	04	Variable-length Honeywell files

The following coding is an example of the macro instructions specializing the $TPPRT conversion program to independent mode, Honeywell (1/2-inch) file format. It specializes the routine to accept variable-length records of up to 136 characters and processes 20 files on the tape. A comma should follow the last parameter used:

```
ØØØØØCD     $TPPRTH,H,,Ø,2Ø,
ØØØ1ØL       15     I,3,2ØØØ,2,3,Ø2,4Ø,1,ABC,,,136,,,,,,Ø4,
```

Further information pertaining to the specialization can be found in the Data Conversion A and C manual, Order Number 231.

The header record of the COBOL D print-image tape contains information describing the type of file which is put out on tape. The COBOL D print-image file is of the following format:

1. Every file has a header record (i.e., a 1HDRΔ)
2. The file name for each file is PRINTΔTAPE.
3. Item length is variable.
4. There is one item per record.
5. The maximum length of each record is 136 characters.
6. Each record is bannered with an octal 57.
7. The file type designation for Data Conversion A and C is 40_8 (i.e., print-image file).
8. There is one vertical format control character for each record.
9. The position in the item for the vertical format control character is 03_8.
10. A count of the number of characters contained in each record precedes each item.
11. An item consists of the three control characters and the actual print image. The first two control characters specify the length of the item (i.e., four less than the number of characters in the record). The third control character is the vertical format variant, which controls the printer spacing for that particular record (the third variant of a printer PDT).

Hence, preceding each actual print image, there are seven octal digits describing the image itself.

If, for any reason, a compilation is not completed and a dump is not taken, the print tape may be terminated by doing a 07 fixed start.

To run Data Conversion A and C against the print tape, the tape must first be rewound. The specialized version of Data Conversion A and C may then be run without interruption or manual entries through the console.

The input tape for COBOL D must be a card-image tape — e.g., a tape created by the SCOPE card-to-tape (for 1/2-inch tapes) program. In this type of five-tape environment, the following rules must be followed:

1. The ECD card must contain a 101 in columns 48 through 50.
2. A printer must be present for output.
3. The card-image tape must be assigned to logical tape unit 1.

There are two ways to begin a compilation using tape input. The first requires the presence of a card reader. In this environment, the user may initiate compilation by using two cards, 1) the Console Call card and 2) the ECD card. All additional information beginning with the visibility card (ABAVP(B)), through the ENDΔCONV or 1EOFΔ card is searched for on the tape. Any number of programs may be on the input tape and may be processed after the 14000 halt. It is unnecessary to resubmit the two input cards.

The tape itself should always contain a header record. In addition, following the last program to be compiled there should be a 1EOFΔ record. When the record containing this card image is read, the tape is cycled down.

If, however, a card reader is not present, it is necessary to initiate compilation with manual entry. In this complete off-line input environment, the following manual entries must be made at the 1702 loader halt:

1. Key 101 into location 100_8 to designate manual entry.
2. Starting at location 104_8, the call name (C)OBOLD (0) 1 (0) must be entered with associated word marks on C, 0, and 0.
3. Starting at location 227_8, the ECD field in the loader, the following must be entered:

Location	Value	Explanation
227	101	Designates tape input
230	00	Trunk assignment
231	01	Tape drive
232	15	See explanation below

The last parameter value, 15 (location 232_8), designates the high bank of memory. This field provides the user with an opportunity to change the high-bank address from that which he specified on the ECD card image on the tape. If he wishes a change in bank size, a bank number (03_8 to 07_8) may be entered and this value overrides the value which is on the tape ECD card image. Otherwise, when no change is desired, the field should be left blank and the bank specified on the tape is the accepted high bank of memory allocation.

When using tape input, the card-image deck on the tape should follow the same format and order as a card input deck. The only additional requirements are that a 1HDRΔ card image precede the deck and that a 1EOFΔ card image terminate all information to be compiled.

The user is cautioned to be aware of the order of programs on the input tape when selecting programs from the MSPT and/or the library tape if these types of compilations are interspersed on the tape. By closely following the order of compilations, the user knows which tape to mount (at 14000 halt) for the next compilation.

Six-Tape Environment

In the six-tape environment, all the rules for both print-image and card-image tapes must be observed with the exception of the changes noted below.

1. The input card-image tape must be assigned to logical tape unit 5.
2. If manual entry is desired, location 231_8 of the ECD field in the loader must contain a 05_8, specifying the input tape as being on logical tape unit 5.

3. The numbers 105 must be punched in the ECD card in columns 60 through 62. (Additional information concerning the ECD card can be found in this section.)

Column	Content	Meaning
6	E	ECD Identification
16-17	00	Low bank address
19-20	03	High bank address (relocatable up to 07)
21-23	100	Address of BRT
	400	Address of drum

i.e. 4th 4K module = 16K. — (in 4K modules.)

Table 8-1. Sample ECD Cards

Running Environment	Card Column														
	5-6	16-17	19-20	21-23	24-26	27-29	30-32	33-35	48-50	51-53	54-56	57-59	60-62	71	72-80
Card Input Printer Output	ΔE	00	03 to 07	100 (400 = Drum)		JJ0	-20	K10		102	103	104			Time and Date
Card Input Print-Image Tape Output Printer	ΔE	00	03 to 07	100		JJ0	-20	K10	101	102	103	104		S	Time and Date
Card Input Print-Image Tape Output Console Typewriter	ΔE	00	03 to 07	100 or 400	570	JJ0		K10	101	102	103	104		S	Time and Date
Card Input Card-Image Tape Input Printer Output	ΔE	00	03 to 07	100 or 400		JJ0	-20	K10	101	102	103	104			Time and Date
Card-Image Tape Input Printer Output	ΔE	00	03 to 07	100 or 400			-20		101	102	103	104			Time and Date
Card-Image Tape Input Print-Image Tape Output Console Typewriter	ΔE	00	03 to 07	100 or 400	570				101	102	103	104	105	S	Time and Date
Card-Image Tape Input Print-Image Tape Output Printer and/or Console Typewriter (full ECD card)	ΔE	00	03 to 07	100 or 400	570	JJ0	-20	K10	101	102	103	104	105	S	Time and Date

Table 8-2. Possible Running Environments

Card Input	Tape Input	Console Typewriter	Printer	Tape Output
*			*	
*			*	*
*		*		*
*		*	*	*
*		*	*	
*	* NOTE		*	
*	* NOTE		*	*
*	* NOTE	*		*
*	* NOTE	*	*	*
*	* NOTE	*	*	
	*		*	
	*		*	*
	*	*		*
	*	*	*	*
	*	*	*	

NOTE: A two-card input deck (i. e. , the Console Call card and the ECD card) may initiate compilation. All additional information starting with the visibility card should be on the card-image tape.

Console Operation:

The operating instructions for the Update program follow those given for compiler operation steps 1 through 11 (see below). When the updating has been accomplished, the input MSPLT (logical 2) should be removed and a work tape mounted for the compilation process. The tapes on logical 3 and logical 4 should be placed in PROTECT status.

COMPILER OPERATING INSTRUCTIONS

Magnetic Tape Unit Set Up:

If an update run has just been performed, the tape on logical 2 must be changed to a work tape; otherwise, the correct tapes are already in place though the status of logical tapes 3 and 4 may have to be changed.

Logical Address	Use	Status
0	Compiler BRT (not relocatable)	PROTECT
2	Work Tape	Permit
3	Work Tape or MSPLT	Permit PROTECT
4	Work Tape or Library Tape	Permit PROTECT

Note that an underlined logical tape address, as above or in the pages that follow, signifies that the tape is relocatable. Note also that where an MSPLT or a Library tape is required it must be replaced by a work tape after the initial phase of the compilation process.

Card Deck Composition:

The first five cards of the COBOL input deck must be the following:

First Card--Must have {COBOLD010 / COBOLH010} punched in columns 1 through 9 and an * in column 18.

Second Card--Equipment Configuration Director Card. (See description of this above.)

Third Card--Must have ABAVPA010 punched in columns 1 through 9 and an * in column 18.

Fourth Card--Must have COBOL*INPUT punched in columns 1 through 11 (for other fields of this card see Section X-14)

Fifth Card--If the source program is to be read through the card reader, this card must have OPTION punched in columns 7 through 12. If the program is to be taken from an MSPLT, SELECT must be punched in columns 7 through 12. (see Section X-12).

The last card in any batch must have ENDΔCONV punched in columns 1 through 8.

The above applies when a compilation run does not follow an MSPLT update. Note that if the compilation immediately follows an Update run the first two cards are not necessary.

NOTE: If both an MSPLT and a Library tape were required, only one halt occurs to allow both messages to be printed. After work tapes have been mounted and the RUN button depressed, the compiler proceeds to completion as previously defined in step 12., above.

15. If it is not desired to execute the object programs immediately after compilation, more compilations can be run by performing the following:

    a. Remove the OBRT on logical tape unit 3.

    b. If the same equipment configuration and time notations as the previous compilation run is desired, remove the {COBOLD / COBOLH} card and the ECD card from the input deck for the next compilation run. Place the remaining cards in the card reader hopper and depress the RUN button. Compilation proceeds as from step 12., above.

    c. If a different equipment configuration or time notation is desired in the next compilation run, rewind the compiler BRT and then depress the RUN button.

Fixed Starts:

To perform any of the following fixed starts, these steps must be performed by the operator:

1. Enter the desired fixed start location 2502_8.
2. Set the sequence counter to location 2522_8.
3. Depress the RUN button.
4. For fixed start 04_8, mount the COBOL D BRT on Logical Tape Drive 0. To execute the object code, place the following three cards in the card reader:

    a. Console call card —

Card column:	1	2	3	4	5	6	7	8	9	18
Content:	A	B	A	O	B	J	0	1	0	*

    b. Drum ECD card - same as compile time drum ECD card.

    c. Visibility card —

Card column:	1	2	3	4	5	6	7	8	9	18
Content:	O	B	J	C	T	|	0	1	0	*

Object Program Visibility

The following are the fixed starts and the actions associated with them.

Start	Action
00_8	Performs a memory dump and does a backwards tape dump, all in octal, of logical tapes 2, 3, and 4. At the end of the dumping process the loader halt (Bc = 14000) is provided. When compiling with drum, only a memory dump is performed. During the execution of an object program, this fixed start results only in a memory dump. After this dump, the equivalent of a 03 fixed start, see below, is automatically performed.

Start	Action
00_8 (cont)	The 00 fixed start is set by the dump program at the beginning of a visibility string and remains in memory location unless another fixed start (except a 01) has been performed. While it is in memory, the fixed start can be performed by setting the sequence counter to 2522_8 and depressing the RUN button.
01_8	Restarts the entire batch compilation from the beginning. By this restart, 2502_8 is set to 00.
02_8	Restarts compilation of the current source program from the beginning of the compilation loop. No resetting of 2502_8 is provided. This fixed start cannot be used at object time.
03_8	Provides for manual tape dumps. A message is displayed on the printer explaining the parameters to be entered by the operator. The parameters are entered in locations 4421_8 through 4425_8. The format of the parameters is nnmct where nn (entered in 4421_8 through 4422_8) is the number of blocks to be dumped. If m, see below, calls for a backward dump, an nn entry of 0000 is interpreted as calling for all blocks to be dumped. m (entered in 4423_8) is the mode: 00 = all done 10 = dump backwards in octal. 20 = dump backwards in alpha. 30 = dump forward in octal 40 = dump forward in alpha. c (entered in 4424_8) is the tape trunk number (normally this is 00). t (entered in 4425_8) is the logical tape address After the desired entries have been made to these locations, depress the RUN button. When the dump is finished, there is a halt provided so that more dumps can be taken if desired. When there are no more dumps to be taken, enter 00 into location 4423_8. The halt resulting from this is the loader halt (B address: 14000). No resetting of 2502_8 is provided.
04_8	Creates a tape and card deck which will be the input to the Drum Load Store program. Drum Load Store program, using the director deck and the tape created here, will load the compiler-generated object code on the drum and allow object code execution from the drum.
	Produces a directory of (1) the system programs necessary to execute

Start	Action
07_8 (cont)	Should only be called when the end of compilation has not been reached and when neither the memory dump program nor the tape dump program has been called.
	Repositions the BRT and comes to the loader halt (B address: 14000) for [illegible]

POSSIBLE ERROR CONDITION.

PROGRAM LIMIT EXCEEDED. THIS UPDATE IS INCOMPLETE.

RESTARTING COMPILATION OF xxxxxx FROM yyyyyy SEGMENT zz.

SET DENSITY ON LOGICAL x TO HIGH AND RUN.

THE REQUEST FOR xxxxxx HAS BEEN IGNORED.

TO IGNORE ERROR - DO START. ELSE FIX AND RESTART RUN.

TO IGNORE ERROR - DO START OTHERWISE FIX AND RELOAD THIS PROGRAM.

TO IGNORE VALIDITY CHECK - DO START OTHERWISE FIX LABEL CARD AND RELOAD THIS PROGRAM.

USABLE - TO COMPLETE LISTING OF THE xxxxxx COMPILATION.

UNABLE TO READ LABEL ON LOGICAL x. CHANGE DENSITY AND RUN.

UNABLE TO xxxxx LABEL ON LOGICAL y. CHANGE TAPE AND RUN.

OBJECT CODE OPERATING INSTRUCTIONS

The object code produced by the compiler is completely self-contained and runs in any desired operating environment. The compiler also produces an operating environment in which the object code can be efficiently and effectively run. Any Series 200 object program, loaded higher than location $2,127_{10}$, can run in this environment. In addition, the compiler produces a small card loader for running in special environments.

If the user does not wish to utilize the provided operating environments, he may use any other Mod I operating environment or create his own. In any case, the COBOL object program performs as directed.

The object code can be executed immediately after the typeout on the printer:

END OF COMPILATION
OBJECT PROGRAM RUN TAPE IS ON LOGICAL TAPE UNIT 3 (OR 2)

The system is then at a loader halt (Bc = 14000). If it is desired to execute the object program at this point, it can only be called by visibility.

Card Deck:

1. The first card of the deck should have OBJCT in columns 1 through 5. The desired visibility in column 6, 010 in columns 7 through 9, and an * in column 18. It can be followed by any necessary data cards.

   As many OBJCT cards as desired may appear with their data in the deck.

Operating Object Programs Immediately after Compilation:

1. Place the card deck in the reader and press START.
2. Mount any necessary tapes.
3. Depress the console RUN button.

4. The first object program with the specified visibility is executed.
5. If the source program had specified SUPERVISOR CONTROL and a halt after loading was not provided, all programs on the OBRT having the specified visibility are loaded and executed.
6. If SUPERVISOR CONTROL was not specified, there is no loader halt after the execution of each object program. There is a program halt (whatever halt the object program contained). The fixed start 11 must be executed in order to obtain a loader halt (Bc = 14000) and continue to run the next object program.
7. If it becomes necessary to dump the object program or to use any of the fixed starts, as explained above, at the conclusion of the specified action, the OBRT returns to a loader halt (Bc = 14000). Depress the RUN button to continue processing.

Operating Object Programs with the OBRT on logical 0:

1. The OBRT is on logical 0 and rewound.
2. Card Deck

   First card — must contain $\begin{Bmatrix}\text{ABBOBJ010}\\ \text{ABAOBJ010}\end{Bmatrix}$ in columns 1 through 9 and an * in column 18.

   Second card — Equipment Configuration Card

   Third card — OBJCT card for tape.

   The rest of the deck is as described under Card Deck, above.
3. The starting of the object programs is now the same as that of the starting of the compile, see Console Operation, 1 through 11, above.

   NOTE: Following step 9 when operating under COBOL H, location 124_8 must be checked for the proper visibility of the appropriate Loader-Monitor. If the visibility is correct, proceed to step 10; otherwise, correct the visibility and proceed to step 10.
4. The execution of the object programs is as described in 4 and 5 of Operating Object Programs Immediately after Compilation, above.
5. If it is desired to call the object program by name rather than visibility, replace the OBJCT card with a card containing the program name in columns 1-6. (Note that this can only be done when the OBRT is on logical 0.)

Relocation of the object tape and peripheral devices at object time can be accomplished by placing an OECD card (see Section VII) immediately after the OBJCT call card regardless of the manner in which the object checkouts were started.

OBJECT RERUN OPERATING PROCEDURES

Rerun Recording Routine:

The Rerun program contains necessary coding to branch to the Rerun Recording Routine

according to what is stated in the RERUN clause. (See Section IV). Whenever the routine is entered, a printout is given on the printer or console typewriter. The typeout is of the following format:

RERUN 0t00 aaaaaa nnnn bb

where

t is the logical address of the tape on which the memory dump is written.

aaaaaa is the program name that identifies a rerun record.

nnnn is the octal identification number of the rerun dump.

bb is the octal number, of which the low-order, four binary bits denote the settings of the sense switches:

01 sense switch 1	04 sense switch 3
02 sense switch 2	10 sense switch 4

After the Rerun Recording Routine is completed, the program continues processing as if no interruption occurred.

Rerun Restoring Routine:

The Rerun Restoring Routine processes the information recorded by the Rerun Recording Routine. It searches a specified tape for the requested Rerun-point file. A console typeout notes the sense switch settings; the tape ID's are checked; and the tapes are positioned. Memory is then restored, and the object program started at the proper point.

In order to RERUN an object program from a rerun-point, the operator must mount the object program tape on logical drive 0 and the reel of tape containing the rerun memory dump on logical tape unit t as indicated in the above typeout.

The following steps must be taken to load a RERUN:

1. Bootstrap the Plus-Tape Loader-Monitor.
2. Obtain halt number 3 (B register = 17002)
3. Enter Search Parameters into the communication area through the console or card reader. The parameters used are the ones obtained at the time the RERUN was recorded.

   a. Card Entry
   Using the above rerun information, place a card with the following format into the card reader and depress the RUN button.

Columns	Contents
1-6	RERUNN
7-8	rerun-point identification number
9	tape drive no. of RERUN tape
10-17	halt name (combined program and rerun-point identification number)
18	*
19-20	not used

b. Console Entry

Again using the previous typeout, data is entered in the following octal locations without disturbing punctuation:

Location	Contents
100	01
104-111	RERUNN
112-113	rerun-point identification number
114	tape drive no. of RERUN tape
115-124	halt name (combined program and rerun-point identification number)

Aftering entering the above information, depress the RUN button. Location 100_8 must be manually reset to octal 00 before a console card can be used after this type entry.

During file-positioning, operator action is requested by the console messages listed below. After the operator performs the last indicated action and depresses the RUN button, normal object program processing resumes.

Output Message	Comments
ffffffff CARDCNT nnnnnnnn	Card reader input where f...f indicates name of card file if it contains standard labels and n...n is the number of cards read at rerun point
ffffffff REEL nnn TAPE t	Tape unit input where f...f indicates file name; nnn, number of reel within file; and t, logical tape drive.
REEL nnn TAPE t	Tape unit without labels.
MULTIPLE REEL TAPE t	Multiple file tape with no file being processed.
PROCESSING OF RERUN IS STARTING	Memory has been restored. Depress the RUN button.

After each of the above messages, the exact number of cards should be inserted in the reader or the proper tape should be mounted on the proper logical tape drive. The RUN button should be depressed after each action is performed.

OVERALL VIEW OF THE COMPILING SYSTEM

As can be seen from the operating descriptions in this section, the Series 200 COBOL Compiling System has been designed to operate completely within the framework of the PLUS system both for compilation and for object program execution. Further, it has been designed so that all processing necessary for efficient installation operation are encompassed within the compiling system itself. Thus, it is possible to create and maintain a library of source programs for compilation, take programs from this library for immediate compilation, load the object programs produced immediately after compilation, and provide immediate checkout of the object program or programs.

The following illustration shows an input card deck to the compiling system in the correct order for complete system operation from library update to compilation to object program loading and checkout.

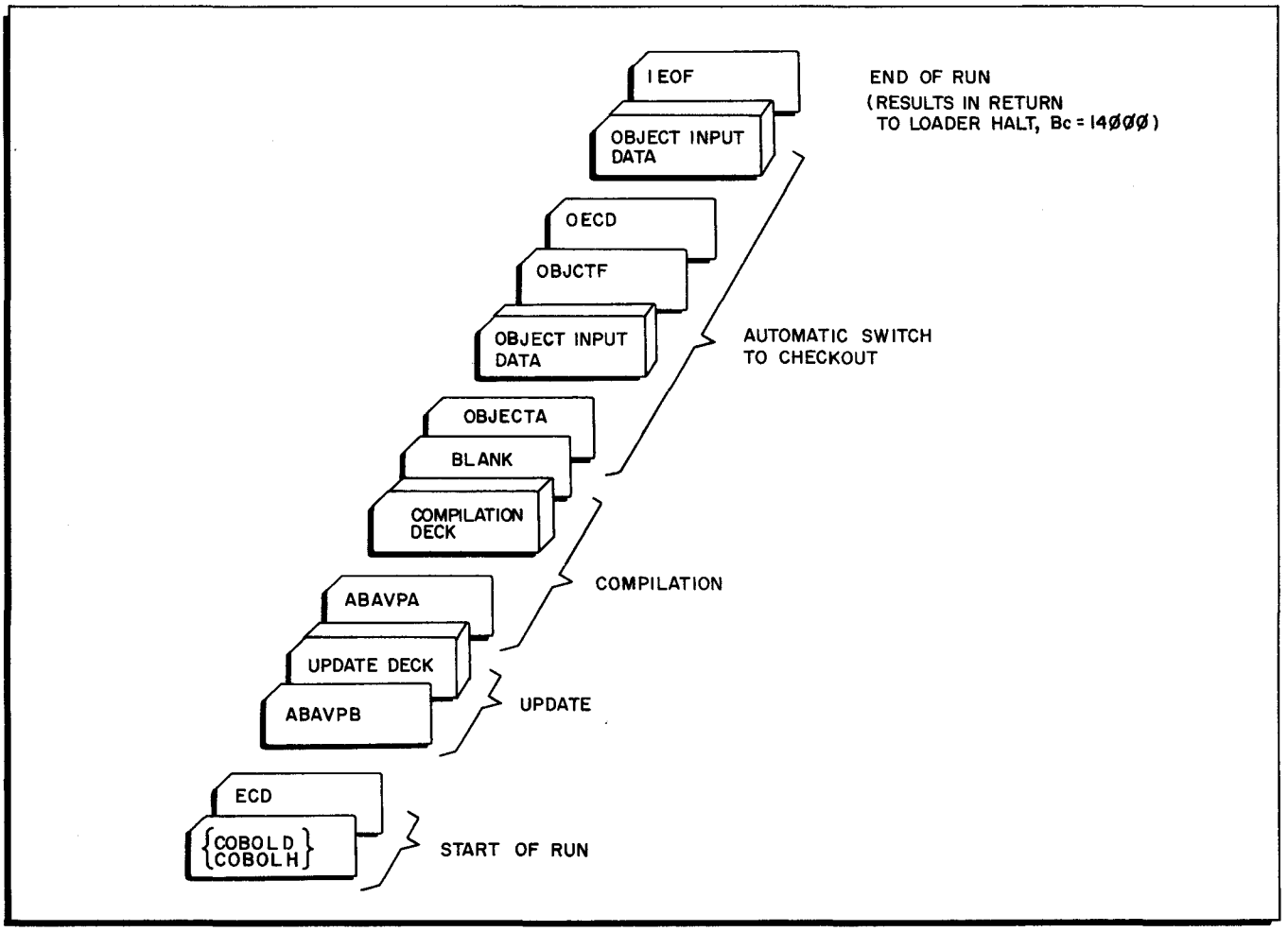


Figure VIII-1. Input Card Deck for Complete System Operation

OPERATIONAL DESCRIPTION

The COBOL D and H Compilers process programs in two phases: the preprocessing phase and the main loop phase. The preprocessing phase prepares a tape for the main phase by resolving the following:

External and internal copies,

Statements containing qualifications,

Renaming clauses,

Clauses utilizing the corresponding option,

References to conditional variables,

Editing clauses, and

Output options specified.

The main loop phase syntactically analyzes the source-language input, generates both I/O and non-I/O code, optimizes this code, and produces the object run tape and compilation documentation.

Table VIII-3 gives a general description of each run of the compilers.

Table VIII-3. Run Functions of COBOL Compilers D and H

Run	Function
Preprocessing Phase	
Run 1	Performs library and maintenance functions.
Run 2	Prepares the source-language tape.
Sort Run	Sorts sections of the PROCEDURE DIVISION in priority order.
Run 13	Determines need for Name-Resolver loop and performs initial resolutions.
Run 14	Resolves all renaming clauses, internal copies, and editing clauses.
Run 15	Initializes tables for resolution of lever 88's, qualifications, and correspondings.
Run 16	Resolves qualifications.
Run 17	Resolves correspondings.
Run 18	Formats Name-Resolver loop output.
Run 19	Final program of Name-Resolver loop that either punches cards, lists programs, or prepares a new source-language tape.
Run 20	Reverses the order of the new source-language tape.
ORT Run	Writes the Loader-Monitor and selected COBOL programs on the object run tape.
Main Loop Phase	
I/O for Runs 3, 4, 5	Contains I/O and syntax interpretor for runs 3, 4, and 5.
Run 3	Performs syntactical analysis of IDENTIFICATION DIVISION and ENVIRONMENT DIVISION.
Run 4	Performs syntactical analysis of DATA DIVISION.
Run 5	Performs syntactical analysis of PROCEDURE DIVISION.
Run 6	Replaces data-names and literals with full item descriptions; places word-mark indicators, initialization values, and source-program constants as object data values.
Run 7	Prepares operands and operators for generation.
Run 7 Exit	Provides premature exit to run 11 if any fatal diagnostics have appeared up through run 7.
Run 8.1	Generates I/O code for object time.
Run 8.2	Generates non-I/O code for object time.
Run 9	Optimizes code and generates same-area punctuation subroutines.

Table VIII-3 (cont). Run Functions of COBOL Compilers D and H

Run	Function
Main Loop Phase (cont)	
Run 10	Produces run tape for object time.
Run 10X	Writes selected nonresident I/O routines on object run tape.
Punch Run	If requested, punches a self-loading deck from object BRT produced during compilation.
Run 11	Produces a sorted diagnostic file for use by Run 12.
Run 12	Produces documentation of the compilation. Control returns to run 3 if there are more programs to be compiled; otherwise, run 12X is called.
Run 12X	Cycles back to the beginning of the compiler tape for another batch of programs or for an end-of-run indication.

SECTION IX

COMPILATION LISTING AND DIAGNOSTICS

General

A listing is produced during compilation, which supplies certain historical and debugging information to the programmer. This listing may be divided into six parts:

1. The first part is an exact listing of the source program. This is actually produced prior to compilation.
2. The second part is the compilation summary and contains external information concerning the particular compilation.
3. The third part is the resolved source program listing which again shows the source program instructions plus the related object program locations for DATA DIVISION entries. Diagnostic messages occur embedded throughout this listing.

   An example of a diagnostic message and its related code follows:

```
          150210   0000     FILE-CONTROL.
          150220   0000          SELECT AFILE ASSIGN TO TAPE-UNIT  HA  TAPE-UNIT  HR
                                     □1
1 OBSERV.                    THIS FILE-NAME IS REPRESENTED IN OBJECT TIME PRINTOUTS AS "FILE 1".  1  3  000101
```

   A lozenge (□) appears beneath the COBOL word in error. A number accompanies the lozenge and relates the error to a diagnostic, which follows the line containing the error.

   The first field of the diagnostic refers to the lozenge number from the last line of code. In this example, the number 1 corresponds to the □1 in the previous line.

   The second field denotes the level of the diagnostic. There are three levels:

   a. OBSERV. - this simply indicates a comment to the programmer and has no effect on compilation.

   b. WARNING - this indicates an error or possible error which does not inhibit object code generation. The compiler makes some assumption and continues.

   c. FATAL - this indicates a serious error which prohibits compilation.

   The third field contains the diagnostic message.

   The fourth field indicates the division in which the error occurred. There are three codes that represent divisions:

   1 - IDENTIFICATION DIVISION or ENVIRONMENT DIVISION.

   2 - DATA DIVISION.

   3 - PROCEDURE DIVISION.

   In this example, the number 1 indicates that the diagnostic occurred in the ENVIRONMENT DIVISION.

The fifth field represents the number of the compiler run in which the error was detected. In this example, the error was diagnosed in Run 3.

The sixth field corresponds to the number of the diagnostic in the diagnostic library.

4. The fourth part lists all subroutines used during object-time, their address and function.

5. The fifth part is the object file table. This includes general information on object files: specifically, table, subroutine, buffer and record addresses, and device assignments for each file.

6. The sixth part contains the actual machine-language produced by the compiler. This is presented in both symbolic format and octal machine-language format. In showing the octal machine-language format of an instruction, the index register column preceding the address portion(s) contains the actual number of the index register. If the address is indirect, this column contains an "I." The absolute memory locations of each segment are also included.

PUNCTUATION NOT FOLLOWED BY A SPACE. PROCESSING CONTINUES AS THOUGH SPACE HAD OCCURRED.

NO SPACE BEFORE BEGINNING QUOTE. PROCESSING CONTINUES AS THOUGH SPACE HAD OCCURRED.

ILLEGAL CHARACTER USED. PROCESSING CONTINUES. CHARACTER ACCEPTED AS LEGAL.

NO SPACE BEFORE LEFT PARENTHESIS. PROCESSING CONTINUES AS THOUGH SPACE HAD OCCURRED.

SPACE PRECEDES RIGHT PARENTHESIS. PROCESSING CONTINUES AS THOUGH SPACE HAD NOT OCCURRED.

WORD LONGER THAN 30 CHARACTERS OR A LITERAL LONGER THAN 120 CHARACTERS. CHARACTERS BEYOND LEGAL LIMIT IGNORED. PROCESSING CONTINUES WITH NEXT WORD.

PUNCTUATION ANALYSIS CANNOT BE PERFORMED ON THIS FILE. INTERNAL TABLES HAVE OVERFLOWED. ANALYSIS CAN BE PERFORMED IF ALL REDEFINITION IS REMOVED.

EXCESSIVE QUALIFICATION.

DUPLICATE SECTION TAG.

COMPILER ERROR. ERROR CODE NOT IN DIAGNOSTIC LIBRARY.

3. ENVIRONMENT DIVISION Diagnostics

SYNTAX ERROR.

THERE SHOULD BE A SPECIFIC DIVISION, SECTION, OR PARAGRAPH-NAME. IT IS EITHER MISSING OR MISSPELLED.

THIS SOURCE-COMPUTER PARAGRAPH IS IN ERROR.

THIS OBJECT-COMPUTER PARAGRAPH IS IN ERROR.

THE MEMORY SIZE SPECIFIED FOR THE OBJECT COMPUTER IS INVALID. THE SOURCE CONFIGURATION WILL BE USED.

MEMORY ASSIGNMENT OVERLAYS THE LOADER. LOWER EXTREME HAS BEEN ADJUSTED ACCORDINGLY.

THE FILE NAMED IN THIS APPLY CLAUSE WAS NAMED IN A PREVIOUS APPLY CLAUSE. THE PREVIOUS APPLY CLAUSE TAKES PRECEDENCE. THIS CLAUSE IS IGNORED.

PROGRAM-ID MISSING.

SEGMENTATION MAY NOT BE SPECIFIED OR IMPLIED WHEN OBJECT PROGRAM IS ASSIGNED TO PUNCH.

SPECIAL RUN IS REQUIRED TO PUNCH THIS OBJECT PROGRAM.

DUPLICATE RERUN STATEMENT. THE FIRST RERUN CLAUSE TAKES PRECEDENCE. THIS CLAUSE IS IGNORED.

THIS RERUN STATEMENT IS IGNORED. THE NAMED FILE IS NOT A LEGAL RERUN FILE.

MULTIPLE FILE USAGE IS ILLEGAL FOR THE NAMED FILE.

THE LAST ENTRY SHOULD END WITH A PERIOD OR THIS IS A SYNTAX ERROR.

THIS SECTION OR PARAGRAPH NAME SHOULD HAVE BEGUN IN COLUMN 8.

SENTENCE OR CLAUSE IS DUPLICATED OR CONTRADICTORY.

THIS FILE IS MULTI-REEL BUT IS NAMED IN A MULTIPLE FILE CLAUSE.

COMPILATION RESUMED WITH THIS SECTION, PARAGRAPH, OR STATEMENT.

REFERENCE TO NON-EXISTENT FILE. FILE NAME EITHER MISSING OR MISSPELLED.

THERE SHOULD BE A SPECIFIC DIVISION, SECTION, OR PARAGRAPH NAME. IT IS EITHER MISSING OR MISSPELLED.

THIS SPECIAL-NAMES PARAGRAPH IS IN ERROR.

SEGMENT LIMIT NOT IN THE RANGE OF 1 THRU 49.

THE NUMBER OF FILES IN THIS PROGRAM EXCEEDS THE LIMITS OF THE INTERNAL TABLES.

THIS SELECT CLAUSE IS IN ERROR.

THIS FILE-NAME DUPLICATES A PREVIOUS FILE-NAME.

THIS I-O-CONTROL PARAGRAPH IS IN ERROR.

NO ALLOWABLE SELECT CLAUSE WAS GIVEN FOR THIS FILE.

THE FILE-NAME IN THIS SAME AREA CLAUSE WAS USED IN A PREVIOUS SAME AREA CLAUSE.

REFERENCE TO NON-EXISTENT FILE. FILE-NAME EITHER MISSING OR MISSPELLED.

SENTENCE OR CLAUSE IS DUPLICATED OR CONTRADICTORY.

ILLEGAL PERIPHERAL DEVICE.

THIS FILE-NAME IS REPRESENTED IN OBJECT TIME PRINTOUTS AS "FILE xx".

NUMBER OF CHARACTERS PER BLOCK SPECIFIED FOR THIS FILE NOT A MULTIPLE OF THE NUMBER OF CHARACTERS IN LARGEST RECORD. BLOCK SIZE WILL BE OPTIMIZED.

POSSIBLE ERROR, MAXIMUM RECORD SIZE FOR THIS CARD FILE EXCEEDS 80 CHARACTERS.

POSSIBLE ERROR, MAXIMUM RECORD SIZE FOR THIS PRINTER FILE EXCEEDS 132 CHARACTERS.

SINCE LABEL RECORDS ARE OMITTED, BEGINNING OR ENDING LABEL PROCEDURE FOR THIS FILE IS IGNORED.

OPEN INPUT IS GIVEN FOR THIS FILE, THUS BEGINNING OR ENDING LABEL PROCEDURE IS CHANGED FROM USE AFTER TO USE BEFORE.

NO OPEN INPUT IS GIVEN FOR THIS FILE THUS, BEGINNING OR ENDING LABEL PROCEDURE IS CHANGED FROM USE BEFORE TO USE AFTER.

FIELD USED AS VALUE OF IDENTIFICATION FOR THIS FILE DOES NOT HAVE A SIZE OF 10. THE VALUE USED IS UNSPECIFIED.

FIELD USED AS DATE-WRITTEN FOR THIS FILE DOES NOT HAVE A SIZE OF 5. THE VALUE USED IS UNSPECIFIED.

FIELD USED AS VALUE OF IDENTIFICATION FOR THIS FILE IS NOT IN WORKING-STORAGE OR THE CONSTANT SECTION.

FIELD USED AS DATE-WRITTEN FOR THIS FILE IS NOT IN WORKING-STORAGE OR THE CONSTANT SECTION.

THIS FILE HAS BEEN SELECTED BUT NOT DESCRIBED IN THE DATA DIVISION AND NOT REFERENCED IN THE PROCEDURE DIVISION.

NO CLOSE WAS GIVEN FOR THIS FILE.

NO OPEN WAS GIVEN FOR THIS FILE.

NO READ OR WRITE WAS GIVEN FOR THIS FILE.

THIS SELECTED FILE, REFERENCED IN THE PROCEDURE DIVISION, IS NOT DESCRIBED IN THE DATA DIVISION. REFERENCED IN THE PROCEDURE DIVISION.

BOTH OPEN INPUT AND OPEN OUTPUT HAVE BEEN GIVEN FOR THIS FILE.

BOTH READ AND WRITE HAVE BEEN GIVEN FOR THIS FILE

ILLEGAL VALUE. VALUE IS IGNORED.

REDEFINES SUBJECT SIZE DISAGREES WITH OBJECT SIZE.

RECORDING MODE MUST BE H-800. H-800 IS ASSUMED.

SYNCHRONIZED DIRECTION MUST BE LEFT. THIS CLAUSE IS IGNORED.

JUSTIFICATION MAY NOT BE SPECIFIED FOR A NUMERIC FIELD. THIS CLAUSE IS IGNORED.

CLASS CLAUSE CONTRADICTS PICTURE CLASS. CLASS CLAUSE IGNORED.

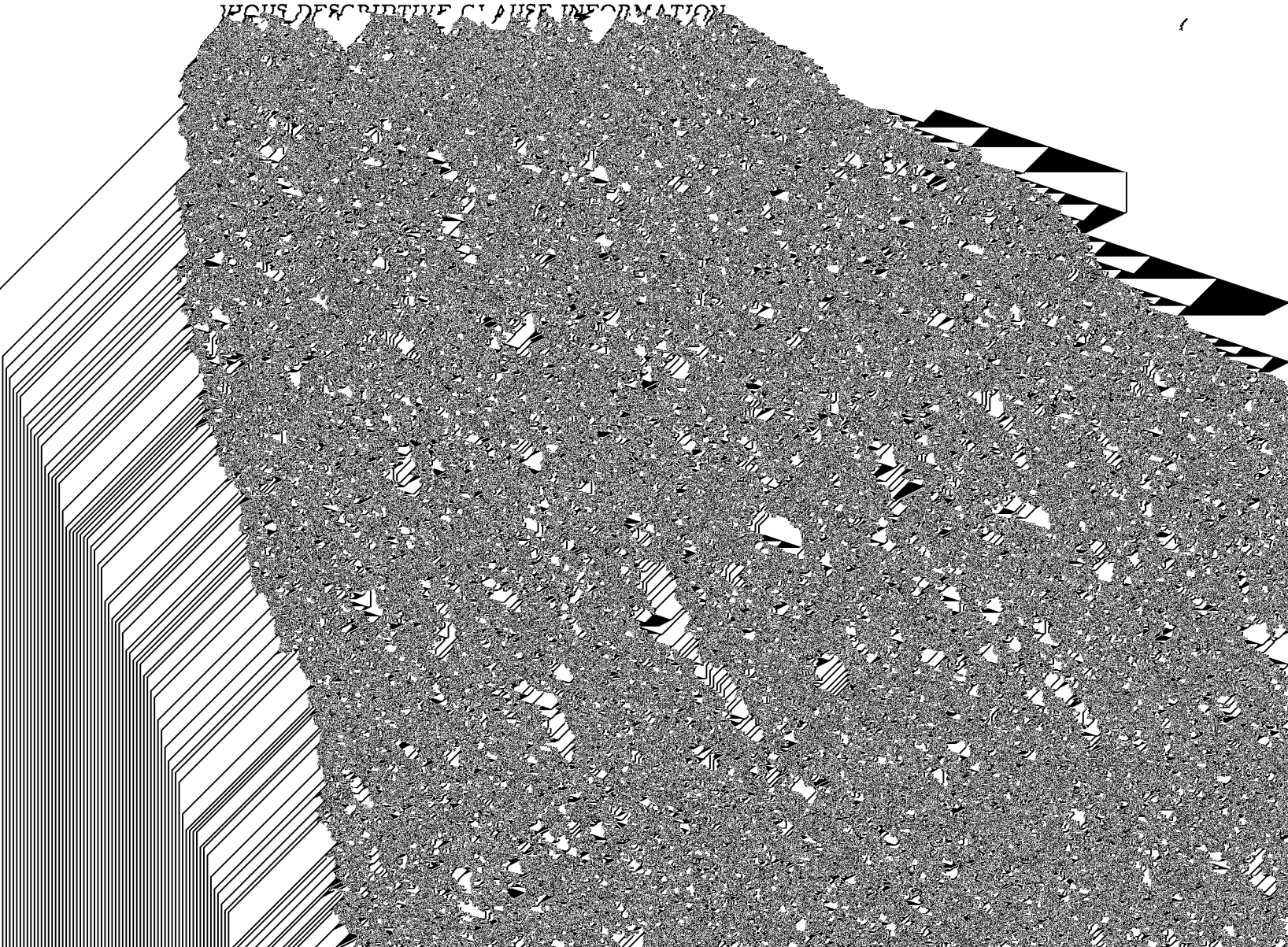
DATA ITEM WITH NON-NUMERIC CLASS MAY NOT BE SIGNED. SIGNED CLAUSE IGNORED.

POINT LOCATION CLAUSE IGNORED BECAUSE THIS DATA ITEM HAS A SIZE GREATER THAN 63 OR HAS A CLASS OTHER THAN NUMERIC.

THE PRECEDING GROUP LEVEL HAS A PICTURE CLAUSE. CLAUSE IS IGNORED.

THIS DESCRIPTIVE CLAUSE IS IGNORED. PREVIOUS PICTURE HAS PROVIDED DESCRIPTION FOR THIS ITEM.

DESCRIPTION PROVIDED BY THIS PICTURE WILL SUPERSEDE PRE-VIOUS DESCRIPTIVE CLAUSE INFORMATION.



TWO CHARACTERS IN THIS PICTURE VIOLATE THE PRECEDENCE RULES.

THIS PICTURE DESCRIBES AND EDITED ITEM CONTAINING NO RECEIVING POSITIONS.

A PARENTHESIZED NUMBER WITHIN THIS PICTURE IS GREATER THAN 4095.

THIS PICTURE CONTAINS AN ILLEGAL CHARACTER.

SIGN IGNORED ON INITIAL VALUE, SINCE FIELD IS UNSIGNED.

POINT ALIGNMENT FORCES LOSS OF SIGNIFICANT CHARACTERS OF INITIAL VALUE.

POINT ALIGNMENT FORCES LOW ORDER TRUNCATION OF INITIAL VALUE.

COMPILER NAME CAPACITY HAS BEEN EXCEEDED, THE NUMBER OF REFERENCED DATA NAMES MUST BE REDUCED.

NO FILE SECTION CARD.

INCORRECT CARD AFTER HEADER CARD.

EVERY RENAMING NOT RESOLVED.

COPY OBJECT NOT FOUND.

INSUFFICIENT QUALIFICATION ON COPY.

LEVEL NUMBERS NOT THE SAME FOR COPY.

SIGNED CLAUSE IS PRESENT WITH EDITING CLAUSES.

GROUP ITEM AND AN EDITING CLAUSE IS PRESENT.

SIZE GREATER THAN 30 AND AN EDITING CLAUSE PRESENT.

DATA-NAME MATCHES A PROCEDURE-NAME.

POINT LOCATION GREATER THAN SIZE, EDITING CLAUSE WILL BE IGNORED.

SYNTAX ERROR IN CONDITION-NAME ENTRY.

TOO MANY CONDITION NAMES, FURTHER CONDITION NAMES WILL BE IGNORED.

NAME-TABLE OVERFLOW.

QUALIFICATION-TABLE OVERFLOW.

EXCESSIVE CORRESPONDING USAGE.

LITERAL SIZE GREATER THAN 63 DECIMAL

ALL SOURCE PROGRAM WORDS BETWEEN THE LAST DIAGNOSTIC AND THIS WORD HAVE BEEN SKIPPED.

SYNTAX ERROR. THIS WORD IS IGNORED.

THIS LITERAL MUST NOT BE GREATER THAN 8 CHARACTERS IN LENGTH.

THERE IS A CONTRADICTION IN THE USE STATEMENT FOR THIS FILE.

THE PRECEDING PROCEDURE TAG HAS APPEARED PREVIOUSLY AS A PROCEDURE TAG.

COMPILER NAME CAPACITY HAS BEEN EXCEEDED. NO MORE NAMES ACCEPTED BEYOND THIS POINT.

IT IS ILLEGAL TO COMPARE TWO LITERALS WITH EACH OTHER.

THIS SUBSCRIPT SHOULD HAVE CLASS NUMERIC.

THIS DATA ITEM SHOULD HAVE CLASS NUMERIC, IF NON-NUMERIC CHARACTERS APPEAR AT OBJECT TIME, RESULTS ARE UNPREDICTABLE.

HIGH ORDER DIGITS WILL BE TRUNCATED AT OBJECT TIME.

LOW ORDER DIGITS WILL BE TRUNCATED AT OBJECT TIME.

THIS DATA ITEM WILL BE TREATED AS AN INTEGER. IF NON-NUMERIC CHARACTERS APPEAR IN IT AT OBJECT TIME, RESULTS ARE UNPREDICTABLE.

DATA ITEM BEING ACCEPTED HAS SIZE GREATER THAN 80 CHARACTERS. ONLY THE FIRST 80 CHARACTERS WILL BE ACCEPTED.

TOTAL SIZE OF ITEM BEING DISPLAYED IS GREATER THAN 120 CHARACTERS. ONLY THE FIRST 120 WILL BE DISPLAYED.

FIELD CAPACITY EXCEEDED.

ILLEGAL TO CLOSE REEL ON MULTI FILE TAPE. STATEMENT ALTERED TO "CLOSE FILE".

THIS DATA NAME WAS NOT DESCRIBED IN THE DATA DIVISION.

TOO MANY DISTINCT SUBSCRIPT EXPRESSIONS WITHIN ONE STATEMENT.

THERE ARE TOO MANY SUBSCRIPTS FOR THE PRECEDING DATA ITEM.

THE PREVIOUS DATA ITEM DID NOT HAVE ENOUGH SUBSCRIPTS.

THE DESCRIPTION OF THIS DATA NAME MAKES IT ILLEGAL FOR USE AS A SUBSCRIPT.

TOO MANY DATA NAMES IN THIS STATEMENT.

THIS PROCEDURE NAME WAS NEVER DEFINED AS A PARAGRAPH OR SECTION NAME.

THIS DATA NAME NOT DESCRIBED IN THE DATA DIVISION.

DESCRIPTION OF THIS DATA ITEM MAKES IT ILLEGAL FOR USE AS AN ARITHMETIC SENDING OPERAND.

DESCRIPTION OF THIS DATA ITEM MAKES IT ILLEGAL FOR USE AS AN ARITHMETIC RECEIVING OPERAND.

ONE OF THE NAMES IN THIS STATEMENT HAS NOT BEEN DESCRIBED AS DATA.

CLASS AND EDITING OF SENDING ITEM CONFLICTS WITH CLASS AND EDITING OF RECEIVING ITEM.

THIS NAME NOT NOT EQUATED TO A HARDWARE DEVICE IN THE ENVIRONMENT DIVISION.

THIS NAME NOT DESCRIBED IN THE DATA DIVISION.

THIS CONDITIONAL STATEMENT INCOMPLETE, THERE IS NO RELATION TEST WORD.

THIS PROCEDURE NOT AN ALTERABLE GO PARAGRAPH.

AN ALGEBRAIC TEST IS BEING PERFORMED ON AN ALPHABETIC ITEM.

FOR ADDRESSING PURPOSES THIS ARRAY MUST BE DESCRIBED LATER FOR THE DATA DIVISION.

INCORRECT LITERAL SUBSCRIPT.

THE CURRENT USE OF THIS NAME CONFLICTS WITH ITS USE ELSEWHERE AS A PROCEDURE NAME.

COMPILER ERROR. ERROR CODE NOT IN DIAGNOSTIC LIBRARY.

NO MATCH IN CVTAB WITH THIS CONDITION NAME. PROBABLY CAUSED BY CVTAB OVERFLOW.

THIS CONDITION NAME REFERENCE IS INCORRECTLY QUALIFIED.

THIS CONDITION NAME REFERENCE IS INSUFFICIENTLY QUALIFIED.

COLUMN EIGHT IS RESERVED FOR PARAGRAPH OR SECTION NAME.

THIS CONDITION NAME WAS DUPLICATED IN THE DATA DIVISION AND MUST BE QUALIFIED HERE.

DATA-NAME IS NOT UNIQUE AND REQUIRES QUALIFICATION.

PROCEDURE NAME NOT UNIQUE AND REQUIRES QUALIFICATION.

PRECEDING DATA-NAME INSUFFICENTLY QUALIFIED.

CORRESPONDING TABLE OVERFLOW, NO MORE CORRESPONDING PHRASES WILL BE RESOLVED.

NAME TABLE OR TABLE OVERFLOW, THE REST OF THE PROCEDURE DIVISION WILL NOT BE RESOLVED.

PRECEDING PROCEDURE-NAME AS QUALIFIED DOES NOT OCCUR IN PROCEDURE DIVISION.

PROCEDURE-NAME REFERENCED DOES NOT OCCUR IN PROCEDURE-DIVISION.

PRECEDING DATA-NAME AS QUALIFIED DOES NOT OCCUR IN DATA-DIVISION.

DUPLICATE PARAGRAPH TAG WITHIN ONE SECTION.

CORRESPONDING DATA-NAME INSUFFICIENTLY QUALIFIED, NON-EXISTANT, OR ITEM-LESS.

SECTION X

SOURCE PROGRAM MAINTENANCE AND SELECTION PHASE PROCESSING

SOURCE PROGRAM AND LIBRARY UPDATE AND MAINTENANCE

This program is a Series 200 COBOL utility program that creates and maintains a Master Source Program and Library Tape (MSPLT). As input, this program processes an input MSPLT and an input Director deck. The Director deck can be read through the card reader or from a card-image magnetic tape. As a result of processing, an output MSPLT and a Library tape are produced. In addition, the programmer can request any or all of the following:

1. A "table of contents" reflecting the organization of the output MSPLT.
2. A list of the program or programs as recorded on the output MSPLT.
3. A punched deck of program or programs as recorded on the output MSPLT.

The processing accomplished by Source Program and Library Update and Maintenance is controlled by Director cards, which are described below. To understand such processing, however, a discussion of MSPLT and Library Tape organization is necessary.

Master Source Program and Library Tape (MSPLT)

The MSPLT contains both source language programs and a library consisting of ENVIRONMENT DIVISION and DATA DIVISION units. Units consist of those sequences of source language lines that can be referenced by

COPY library-name FROM LIBRARY

statements in a source program.

Basically, an MSPLT is a multi-file reel. The files (and the order in which they appear on the magnetic tape) are:

1. Source Program file.
2. ENVIRONMENT DIVISION Library Unit file.
3. DATA DIVISION Library Unit file.
4. PROCEDURE DIVISION file.

In the creation of an output MSPLT, the order of programs and units within their respective files adheres to the following rules:

1. When there is no input MSPLT, the order is that of the programs and units within the Director deck.
2. When there is an input MSPLT, the order of programs and units on the output MSPLT follows input MSPLT order. Programs and units can be inserted within this order from the Director deck (in Director deck order).

Library Tape

The Library Tape produced by this Update and Select program is a copy of the last three files from the output MSPLT. In the creation of the Library Tape, a "dummy" source program file is placed first. This is followed, in tape order, by:

The ENVIRONMENT DIVISION file.

The DATA DIVISION file.

The PROCEDURE DIVISION file.

The Library Tape is created to minimize tape passing during the first phase of compilation, which is the point at which COPY library-name statements are replaced by the appropriate library units.

Director Cards

The Director cards that control processing can be divided into two types:

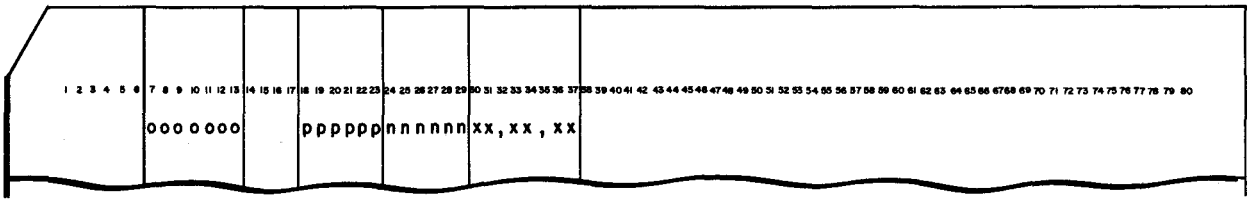
Program Directors – to determine the action to be taken concerning specific programs.

Sentinel Directors – to signal the beginning of the Director deck, the MSPLT file to be processed at a particular time, and the end of the Director deck.

It is in the above order that the Director cards are discussed.

Program Director Cards

General Card Format:



where:

ooooooo (columns 7 through 13) is the operation to be performed. This entry must be one of the following:

ERASEΔΔ	SELECTΔ
INSERTΔ	DELETEΔ
REPLACE	OPTIONΔ
CORRECT	

Each of these is discussed below.

pppppp (columns 18 through 23) is the name of a source program or library unit on the input MSPLT. This entry is left justified in the card field and cannot contain any imbedded blanks. Trailing blanks are allowed when the name contains fewer than six characters.

nnnnnn (columns 24 through 29) is the name to be assigned to a source program or library unit when it is placed on the output MSPLT; this name must be left-justified in the card field and cannot contain any imbedded blanks. Trailing blanks are allowed when the name contain fewer than six characters.

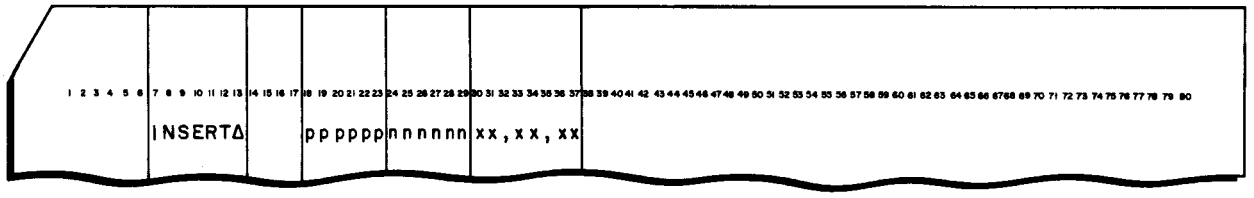
xx, xx, xx (columns 30 through 37) designates optional action(s) that may be desired by the programmer. Each optional action is specified by one of the following codes:

All other card columns are left blank.

2. INSERT

Purpose: To cause a source language program or library unit contained on punched cards to be placed on the output MSPLT as a new program or unit.

Card Format:



where:

INSERTΔ is always entered in columns 7 through 13.

pppppp (columns 18 through 23) is the name of the program or unit which is to precede the new program or unit on the output MSPLT. If this entry is not made, the new program is placed after the last program (if any) that was placed on the output MSPLT.

nnnnnn (columns 24 through 29) is the name to be assigned to the new program when it is placed on the output MSPLT.

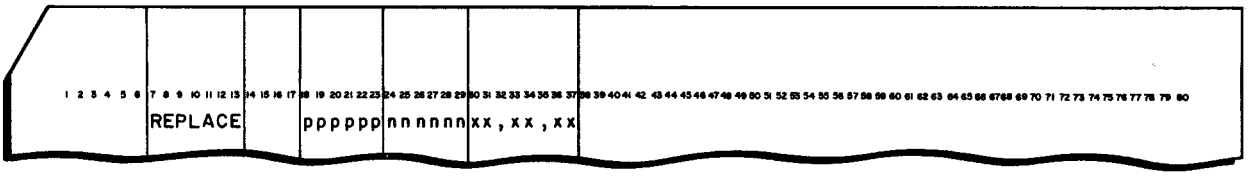
xx, xx, xx (columns 30 through 37) specifies the optional action(s) desired by the programmer (see the General Card Format, above).

All other card columns are left blank.

3. REPLACE

Purpose: To erase a source program or a library unit from the MSPLT and replace it with a source program or a library unit contained on punched cards.

Card Format:



where:

REPLACE is always entered in card columns 7 through 13.

pppppp (columns 18 through 23) is the name of the source program or library unit on the input MSPLT that is to be replaced.

nnnnnn (columns 24 through 29) is the name to be assigned to the program or unit that replaces pppppp.

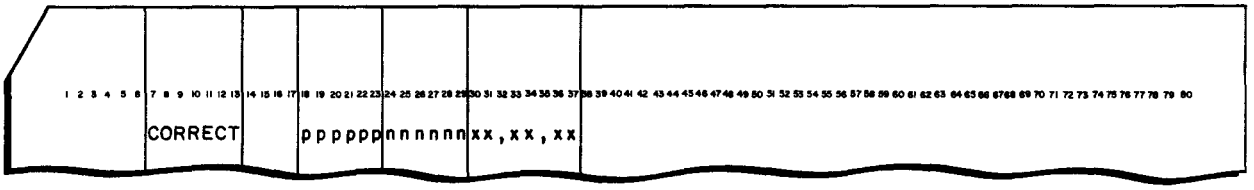
xx, xx, xx (columns 30 through 37) specifies the optional action(s) desired by the programmer (see the General Card Format, above).

All other card columns are left blank.

4. CORRECT

Purpose: To correct source program or library unit lines by replacement, deletion, or insertion of specified lines (according to the cards that follow this director).

Card Format:



where:

CORRECT is always entered in card columns 7 through 13.

pppppp (columns 18 through 23) is the input MSPLT program or unit to have line correction made to it.

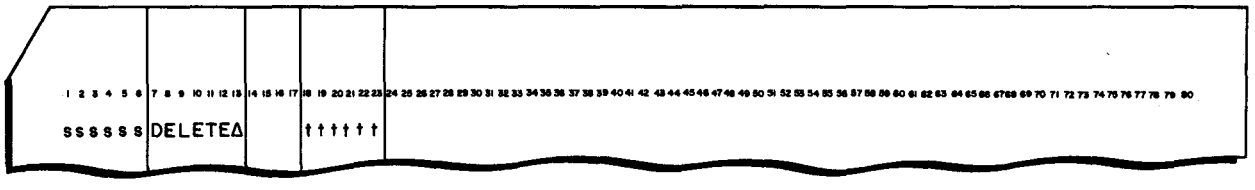
nnnnnn (columns 24 through 29) is the name to be assigned to pppppp when it is placed on the output MSPLT. If this entry is not made, the name on the output MSPLT is pppppp.

xx, xx, xx (columns 30 through 37) specifies the optional action(s) desired by the programmer (see the General Card Format, above).

The method of correction is by means of the following cards.

a. DELETE Source Line

Purpose: To cause the deletion of a source language line (or sequence of lines) from an MSPLT program or unit before the program or unit is placed on the output MSPLT.



where:

DELETEΔ is always entered in card columns 7 through 13.

ssssss (columns 1 through 6) is the sequence number of the source line on the input MSPLT that is to be deleted. If there is a tttttt entry, see below, ssssss is the number of the first line in a contiguous sequence of source lines that are to be deleted.

xx, xx, xx (columns 30 through 37) specifies the optional action(s) desired by the programmer (see General Card Format, above).

See the previous discussion on "Method of Correction."

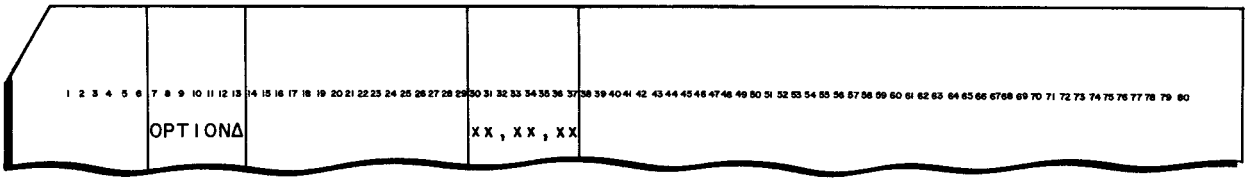
All other card columns are left blank.

Note that if neither a PN nor an LN option is specified, the use of a SELECT director is superfluous since the program or unit pppppp would have been copied from the input to the output MSPLT automatically.

6. OPTION

Purpose: To cause source language cards following this director to be punched out or listed, if requested. No reference is made to the input MSPLT and the source language cards so punched out do not become a part of the output MSPLT.

Card Format:



where:

OPTIONΔ is always entered in card columns 7 through 13.

xx, xx, xx (columns 30 through 37) specifies the optional action(s) desired by the programmer (see General Card Format, above).

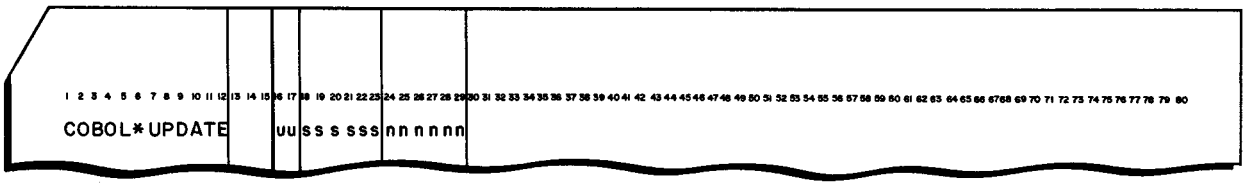
All other card columns are left blank.

Sentinel Director Cards

1. COBOL*UPDATE

Purpose: To mark the beginning of the input Director deck. If any cards occur before this sentinel, they are ignored.

Card Format:



where:

COBOL*UPDATE is always entered in card columns 1 through 12.

uu (columns 16 through 17) specifies the type of update to be accomplished and must be one of the following entries:

ΔΔ Normal update utilizing an input MSPLT and producing an output MSPLT and a Library tape. The version numbers of the output tapes produced is equal to the version number of the input MSPLT + 1.

NΔ A new series of MSPLT's is to be produced. A dummy input MSPLT with a version number of 0 is assumed. An output MSPLT and a Library tape are to be produced, both with a version number of 1.

RΔ The same action as though an entry of ΔΔ had been made except that the output tapes are to have version numbers of 1.

LΔ The same action as though an entry of ΔΔ had been made except that at the conclusion of the run a table of contents of the output MSPLT is to be listed.

RL A combination of the actions produced by RΔ and LΔ entries.

NL A combination of the actions described for the NΔ and the LΔ entries.

Note: It is strongly recommended that one of the L options always be given so that a current table of contents for the last produced output MSPLT is available.

ssssss (columns 18 through 23) is the series name of the input MSPLT. This entry must be made unless uu is either NΔ or NL.

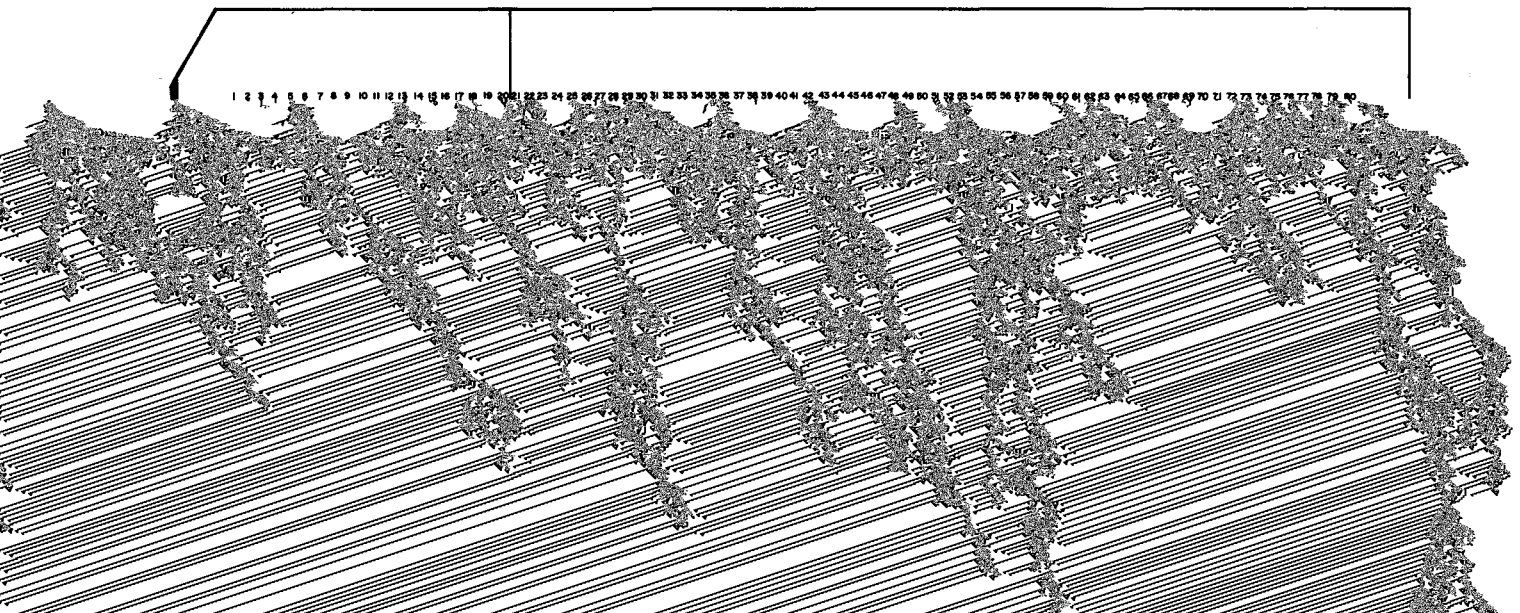
nnnnnn (columns 24 through 29) is the series name to be assigned to the output tapes when uu is NΔ or NL. If neither NΔ nor NL is given, this becomes a request to change the series name from that appearing on the input MSPLT (only the new name appears in this case on the output MSPLT and the Library tape). If nnnnnn = ssssss, this entry need not be made.

All other card columns are left blank.

2. Library Unit File Sentinel

Purpose: To signal that the cards following until the next sentinel are to apply to a particular library unit file.

Card Format:

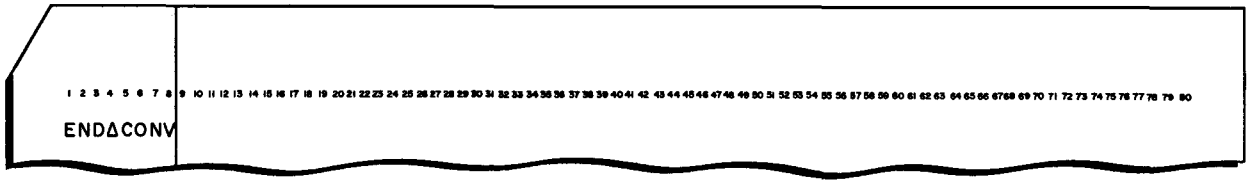


All other card columns are left blank.

3. ENDΔCONV

Purpose: To signal the end of the input Director deck for the run.

Card Format:



where:

ENDΔCONV is always entered in card columns 1 through 8.

All other card columns are left blank.

Director Deck Organization

The Director deck must follow the order of the input MSPLT by:

1. File (i.e., Source Program, ENVIRONMENT DIVISION, DATA DIVISION, PROCEDURE DIVISION).
2. Program (or unit) within file.
3. Sequence number within program. (Note that sequence numbers must be in ascending order within any given program and that when replacement or deletion of lines is called for, the sequence numbers of the correction and deletion cards must be in ascending sequence.)

Once the Director deck has been ordered, the cards necessary for PLUS system operation are added to the beginning and the end of the deck and the run can then be made. The following diagram illustrates a "speed-deck" set up for a Source Program and Library Update and Select run.

Run Processing

After performing any necessary tape label processing functions, as called for by the type of update specified on the COBOL*UPDATE card, processing to produce the output MSPLT from the input MSPLT and the input card deck is begun.

Programs that are not mentioned on Program Director cards are copied unchanged to the output MSPLT. All other programs are subject to the action specified by the Director card on which they are named. As directed, new programs or library units are inserted on the output MSPLT in the requested positions.

When a Library Unit File sentinel is encountered, the input MSPLT is copied to the output

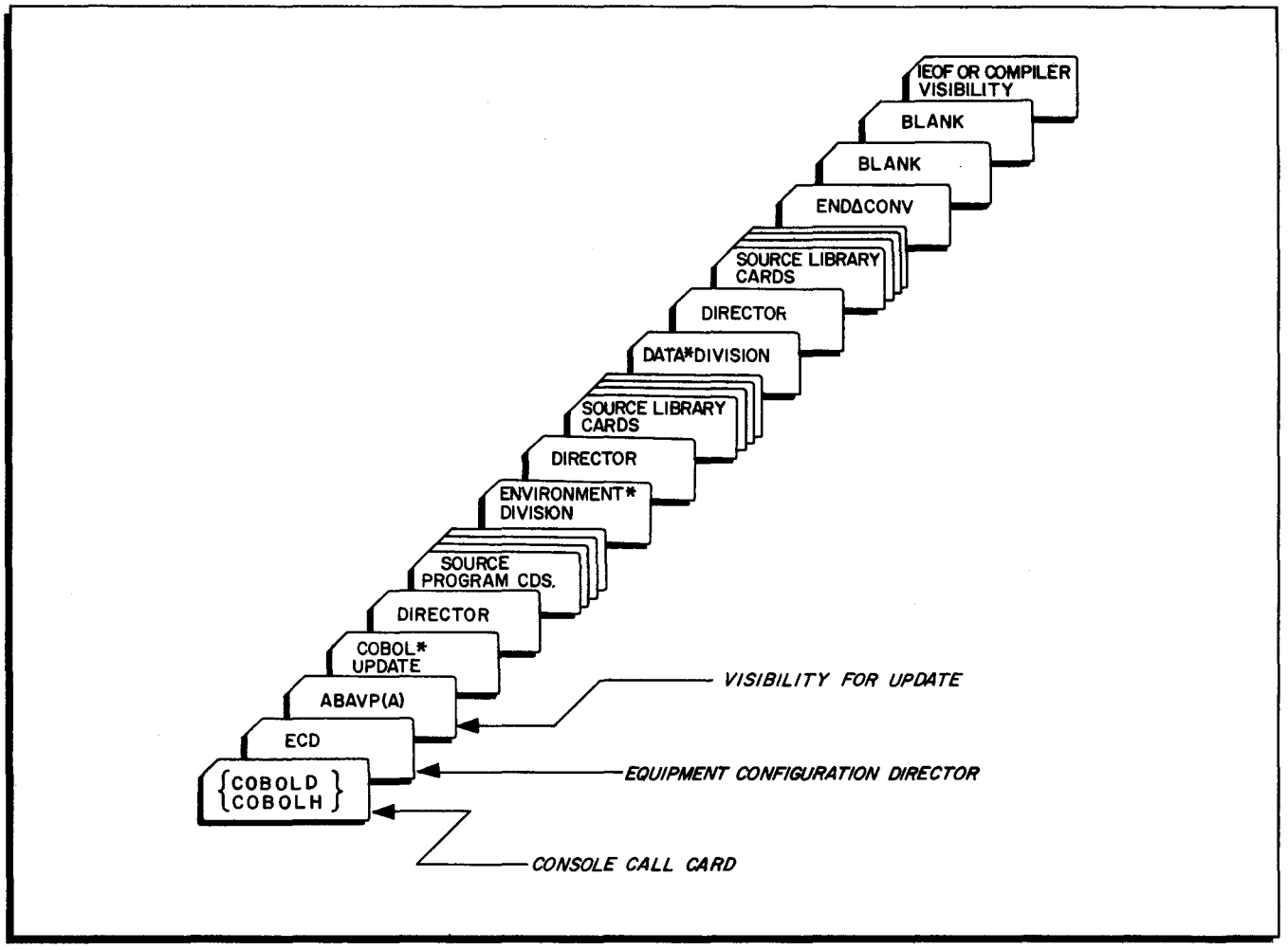


Figure X-1. "Speed-Deck" for Source Program and Library Update and Select Run

MSPLT up to the beginning of the named file. In addition, at the time the ENVIRONMENT DIVISION Library Unit file is encountered on the input MSPLT, this file and the remaining files are written on both the output MSPLT and the Library tape in their updated format (note that only an output MSPLT is produced up to the time that the ENVIRONMENT DIVISION Library file is found).

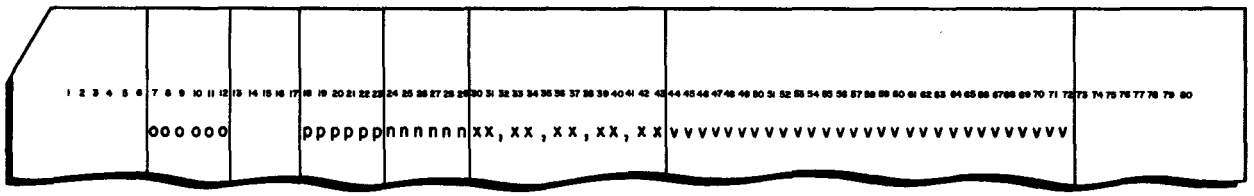
This copy and update loop continues until the END CONV Director is encountered. The remainder of the input MSPLT is then copied, and a table of contents is printed if a request for one has been made. The input MSPLT is rewound with interlock. The output MSPLT and the Library tape are rewound in preparation for possible immediate use as input to compilation.

During the course of processing, the following actions are taken in regard to source program sequence numbers. If the first card of a source program has a blank sequence number field (or if the RN option has been specified for the program), no sequence number check is

made. Sequence numbers are assigned during the processing of the program. The first number assigned is 000010, and each succeeding line number is assigned by the addition of 000010 to the last assigned number. In the case where a sequence check is made (the first card had an entry in the sequence number field and RN was not specified), the occurrence of a blank sequence number field or of any sequence number less than the previous sequence number causes an END

Program Director Cards

General Card Format:



where:

ooooo (columns 7 through 12) is the operation, either OPTION or SELECT, to be performed. (Both OPTION and SELECT are discussed below.)

pppppp (columns 18 through 23) is the name of a program on the Master Source Program Tape (MSPT) that is to be compiled.

nnnnnn (columns 24 through 29) is either the name of a program on punched cards that is to be compiled or a name to be assigned to an MSPT program for this compilation run only.

xx, . . . , xx (columns 30 through 43) designates optional action(s) that may be desired by the programmer. Each optional action is specified by one of the following two letter codes:

RN-Renumber the sequence numbers of the source program.

IA-Accept the order in which the source language lines appear as the correct order.

LN-Prepare a listing of the source program in its "non-resolved" state.

PR-Prepare a punched-deck in its "resolved" state.

SO-Suppress object listing.

DO-Compile for diagnostics only.

LR-Prepare a listing of the source program in its "resolved" state.

CP-The source program is to be compiled and a listing of it in its "resolved" state prepared.

Note that CP, the compiling option, calls for the preparation of a "resolved" listing (the LR option). Because of this, the use of both CP and LR on the same Program Director card is redundant.

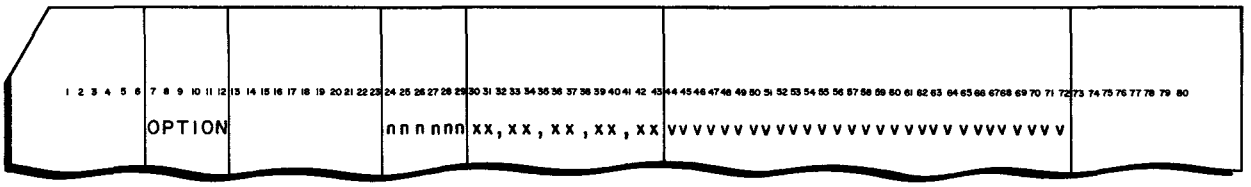
When options are specified, the first entry must begin in column 30. Each two letter code is separated from the preceding two letter code by a comma. There can be no imbedded blanks; however, unused character positions from the last character of the last two letter code through card column 43 are left blank.

vv...vv (columns 44 through 72) is the visibility code or codes to be associated with the object program resulting from compilation of the source program. This entry should only be used when the CP optional action has been specified in card column 30 through 43. If left blank, this program will be visible to all visibilities.

All other columns are left blank.

1. OPTION

Purpose: To cause a source language program contained on punched cards to be accepted for compilation.



where:

OPTION is always entered in columns 7 through 12.

nnnnnn (columns 24-29) is the name of the source program on punched cards which follows this director card in the input deck.

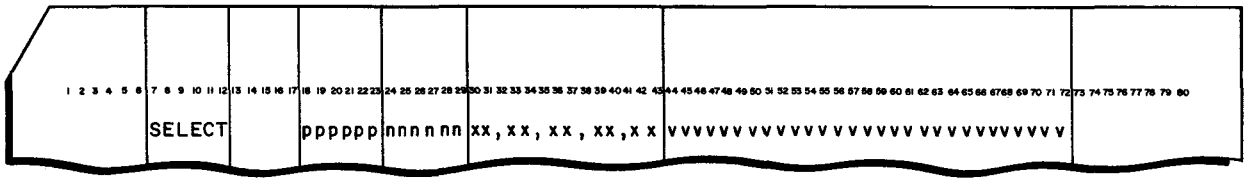
xx,..,xx (columns 30 through 43) specifies optional action(s) desired by the programmer (see the General Card Format, above).

vv...vv (columns 44 through 72) is the visibility code(s) to be associated with the object program produced by compilation of nnnnnn. This entry is made only if CP was designated as an optional action.

All other columns are left blank.

2. SELECT

Purpose: To cause a source language program contained on the MSPT to be accepted for compilation (with or without source program corrections).



where:

SELECT is always entered in card columns 7 through 12.

pppppp (columns 18 through 23) is the name of the MSPT program to be selected.

nnnnnn (columns 24 through 29) is the name to be assigned to the program from the MSPT for this compilation only. (If this entry is not made, the program is compiled under the name it has as it exists on the MSPT.)

xx,...,xx (columns 30 through 43) specifies optional action(s) desired by the programmer (see the General Card Format, above).

vv...vv (columns 44 through 72) is the visibility code(s) to be associated with the object program produced by the compilation of pppppp. This entry is made only if CP was designated as an optional action.

All other columns are left blank.

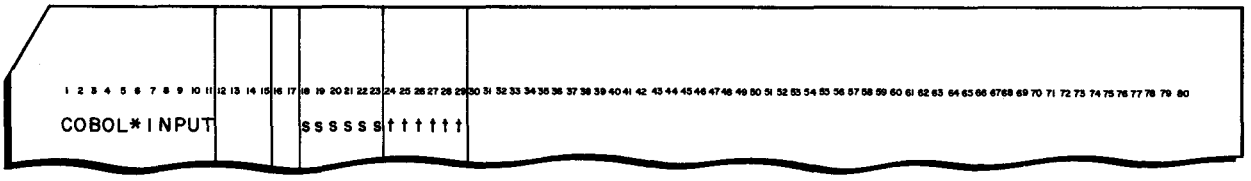
Source language correction cards can follow the SELECT Director. Corrections and insertions are made by line number; that is, a source language line in the MSPT program is replaced by a source language line from a card containing the same line number. Correction line cards must be in the same order as the lines they are to replace in the MSPT program. Such corrections are good only for the particular compilation run. They do not affect the program as it exists on the MSPT.

Sentinel Director Cards

1. COBOL*INPUT

Purpose: To mark the beginning of the input card deck to Selection Phase processing. If any cards occur before this sentinel, they are ignored.

Card Format:



where:

COBOL*INPUT is always punched in columns 1 through 11.

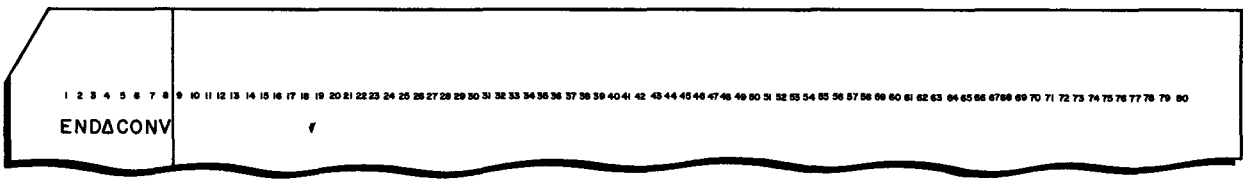
ssssss (columns 18 through 23) is the name of the MSPT series if this is to be checked. If no check is desired, this field should be left blank.

tttttt (columns 24 through 29) is the Library Tape series name if this is to be checked. If such a check is not wanted, this field should be left blank.

All other columns are left blank.

2. END CONV

Purpose: To mark the end of the input card deck to Selection phase processing. No cards following this sentinel are processed.



where:

ENDΔCONV is always punched in columns 1 through 8.

All other columns are left blank.

Organization of Input Card Deck to Selection Phase

The assumption of the Selection Phase is that the input deck it is to process and that directs its processing is a continuous array of punched cards starting with the COBOL*INPUT Director card and ending with the ENDΔCONV Director card. Between these two cards are the desired Program Director cards and source language cards as necessary.

The order of the Program Director and source language cards must follow the following rules:

1. The source language cards comprising the program named on an OPTION Director card must immediately follow that OPTION card.
2. Source language cards that are to correct an MSPT program named on a SELECT Director card must immediately follow that SELECT card. In addition, any such correction cards must be in the order of the lines they are to correct by line sequence number.
3. Within the Selection Phase input card deck, all SELECT Director cards must be in MSPT order.

Selection Phase Processing

After any tape label checking (of the MSPT or the Library Tape or both if called for by entries on the COBOL*INPUT card), the Selection phase reads the Director deck and submits each program for processing from either the MSPT or from the input deck. If necessary, corrections from the input deck are applied to programs from the MSPT. Processing then consists of scanning the submitted program and resolving any "external" COPYs. Thus, when "COPY from library" is encountered, a search is made on the Library Tape for the library entry to be copied. The entry is substituted for the COPY clause. This processing continues on a program by program basis until the ENDΔCONV card is encountered. At time time, end blocks are written on the Scanned Output tape and any input tapes (MSPT, Library Tape) are rewound with interlock.

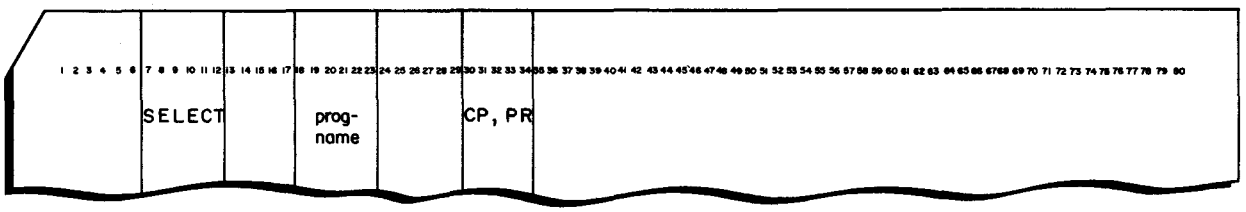
SECTION XI
LANGUAGE PREPROCESSOR

(increases compilation time)

The Language Preprocessor, through qualification, allows the use of duplicate names within a program (see "Qualifying" in Section I). The preprocessor also gives the programmer additional COBOL language elements. These elements are: RENAMING, internal COPYs, editing clauses (ZERO SUPPRESS, CHECK PROTECT, FLOAT DOLLAR SIGN, LEAVING PLACES), level 88's, conditional-variable test, and the CORRESPONDING options to ADD, SUBTRACT, and MOVE.

The preprocessor operates as an optional pre-compilation pass and resolves the above language elements into elements which the COBOL compiler will accept. In this way, time-consuming processing is accomplished outside the compiler. The language elements involving the preprocessor are presented here so that the user realizes that this processing is apart from the compiler and that, if he uses these elements to gain more COBOL language power, he will be taxed an additional amount of compile time.

In most cases, the Language Preprocessor need only be used at the initial compilation. At this time, the programmer should request a punch-resolved deck and use this deck for future compilations. The user may obtain this deck by inserting the following card:



See Section X for a complete description of this Program Director card.

If, at any subsequent time, the user desired to change or add a statement involving a preprocessor element, he must again enter the Language Preprocessor pass.

RENAMING

FUNCTION: To duplicate a file description.

FORMAT:

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL. SELECT [OPTIONAL] file-name-1 [RENAMING file-name-2] ...

TECHNICAL NOTE:

If the RENAMING option is used, file-name-1 utilizes the file de-

internal COPY

FUNCTION: To duplicate within this record all or part of another Record Description contained in a library.

FORMAT:

DATA DIVISION.
.
.
.
level-number data-name-1 [REDEFINES...] ; COPY data-name-2 [FROM LIBRARY] .

TECHNICAL NOTES:

1. When the information to be duplicated is in the source program, the duplication process replaces the COPY clause by the clauses (if any) that follow the data-name in the data-name-2 entry. The duplication process then inserts following the data-name-1 entry, all record description entries that are subordinate to the data-name-2 entry, i.e., up to but excluding the appearance of an entry whose level-number is equal to or less than the level-number of the data-name-2 entry. If the level-number of data-name-1 is 77, data-name-2 must be an elementary item.
2. There is no level number adjustment. The level numbers of data-name-1 and data-name-2 must be identical.
3. When the information to be duplicated is in the source program, the record description entry for data-name-2 may appear either before or after that for data-name-1.
4. When the information to be duplicated is in the library, the additional option FROM LIBRARY must be included. If the FROM LIBRARY option is used, data-name-2 must be the name given to an item in the DATA-FILE portion of the library. Only entire items may be COPYed. If data-name-1 is an elementary item, the library entry must not begin with a level number and a data-name, because both the level number and the data-name would be COPYed, not replaced.
5. When a SELECTed file-name appears as file-name-1 in the RENAMING clause, the file-name must not be used as a qualifier in any COPY statements. Instead, the file-name which appears as file-name-2 in the RENAMING clause can be used, because it has the same structure.
6. The level-number of data-name-1 may not be 88.

→ Select File 1 Renaming File 2 Assign To tape Unit HB (File 1 is copied from File 2! File 2 is the original File
Coulant says "Copy File 1." Would have to say: "Copy File 2".

Editing Clauses

FUNCTION: To permit suppression of non-significant zeros and commas, and to permit floating dollar signs or check protection.

FORMAT:

```
DATA DIVISION.
.
.
.
level-number...  [ ; { ZERO SUPPRESS          }  [LEAVING integer-4 PLACES] ]
                     { CHECK PROTECT          }
                     { FLOAT DOLLAR SIGN      }
```

TECHNICAL NOTES:

1. The editing clauses can be specified only at the elementary item level and only for fixed-length items.
2. Only one editing clause can be given per item.
3. The rules for editing, as shown in the MOVE verb, specify that data items are moved in conformity with the Record Description of the receiving item.
4. The three options, ZERO SUPPRESS, CHECK PROTECT, and FLOAT DOLLAR SIGN, all permit suppression of leading zeros and commas, if the LEAVING option is not employed, suppression stops as soon as either a non-zero digit or the decimal point (actual or assumed) is encountered. Specifically:
   a. When ZERO SUPPRESS is specified, leading zeros and commas are replaced by spaces.
   b. CHECK PROTECT is not a Language Preprocessor element but is included here, because it is one of the standard COBOL editing clauses (see Section V).
   c. When FLOAT DOLLAR SIGN is specified, the rightmost character suppressed is replaced by a dollar sign, and all other characters which are suppressed are replaced by spaces.
5. The LEAVING option can be employed to stop suppression before the decimal point (actual or assumed) is encountered. Integer-4 must be a numeric literal with an integral value. When used, suppression stops at the character just prior to the integer-4 position, unless stopped sooner by the rules specified in 4 above. The integer 4 position is a count of the number of characters starting immediately at the left of the actual or assumed decimal point.
6. Because dollar signs and asterisks are alphanumeric, those options cannot be specified unless the item meets the specifications of a standard data format.
7. More comprehensive editing features are available in the PICTURE clause. When any of the above options are used, the format of the item is assumed to contain editing symbols.

Condition-Name Entry
(Level 88)

FORMAT:

DATA DIVISION.
FILE SECTION. or WORKING-STORAGE SECTION.

nn data-name
88 condition-name

TECHNICAL NOTES:

1. Each condition-name requires a separate entry with level numer 88. This entry contains the name of the condition and the value, values, or range of values associated with the condition-name. The condition-name entries for a particular conditional variable must follow the entry describing the item with which the condition-name is associated. A condition-name can be associated with any elementary or group item except the following:

   a. another condition-name;

   b. a group containing items requiring separate handling due to synchronization, USAGE, etc.

   88's are not allowed in the CONSTANT SECTION.

2. Condition-Name Entry Format

   Some of the possible ways of writing condition-name entries are:
   nn data-name

```
88 condition-name-1 VALUE IS literal-1..
88 condition-name-2 VALUES ARE literal-2, literal-3
88 condition-name-3 VALUES ARE literal-4(AND)literal-5(AND)..
88 condition-name-4 VALUES ARE literal-6 THRU literal-7.
```

   As an example:

```
03 GRADE SIZE IS 2 CHARACTERS NUMERIC.
   88 PRIME VALUE IS 1.
   88 SECOND VALUE IS 2.
   .
   .
   .
   88 GRADE-SCHOOL VALUES ARE 1 THRU 6.
   88 JUNIOR-HIGH VALUES ARE 7 THRU 9.
   88 HIGH-SCHOOL VALUES ARE 10 THRU 12.
   88 GRADE-ERROR VALUES ARE 0, AND 13 THRU 99.
```

condition-name condition

FORMAT:

PROCEDURE DIVISION.

paragraph-name.

IF [NOT] condition-name

TECHNICAL NOTES:

1. In a condition-name condition, a conditional variable is tested to determine whether or not its value is equal to one of the values or lies within a range of values associated with a condition-name.

2. The condition-name is the name of a level 88 entry in the data division. The result of the test is true if the literal value corresponding to the condition-name is present in the elementary item preceding the description of the condition-name in the data division.

3. Additional information concerning conditional statements may be found in Section VI, under the IF statement.

ADD CORRESPONDING

FUNCTION: To add series of numeric data items through the performance of multiple operations.

FORMAT:

PROCEDURE DIVISION.

paragraph-name.

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

[ROUNDED] [; ON SIZE ERROR any imperative statement]

TECHNICAL NOTES:

1. Object coding produced: for each separate addition and storage of items, coding is generated as though an ADD verb statement had been written for each; however, the SIZE ERROR coding generated is only for the last pair of operands ADDed.
2. If the CORRESPONDING option is used, multiple operations are performed. The operations are defined by matching the data-names of numeric elementary items subordinate in hierarchy to data-name-1 and data-name-2. Data-names match if they and all their possible qualifiers up to, but not including, data-name-1 and data-name-2 are the same.
3. When the CORRESPONDING option is used. Only the first complete description of any area is considered in the case where a REDEFINES has been used. All items describing the data contained within a REDEFINES group are ignored.
4. If the CORRESPONDING option is used, no items in the group referred to can contain an OCCURS clause.
5. In the CORRESPONDING option, data-name-1 and data-name-2 must not have a level number of 77 or 88.
6. For a discussion of the general rules regarding ADD, see Section VI.

MOVE CORRESPONDING

FUNCTION: To transfer data, in accordance with editing rules, to one or more data areas.

FORMAT:

PROCEDURE DIVISION.

paragraph-name.

MOVE CORRESPONDING data-name-1 TO data-name-2 [,data-name-3...]

TECHNICAL NOTES:

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

SUBTRACT CORRESPONDING

FUNCTION: To subtract one or more numeric data item(s) from a corresponding item and set the value of an item equal to the result.

FORMAT:

```
PROCEDURE DIVISION.

paragraph-name.

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

   [ROUNDED]    [; ON SIZE ERROR any imperative statement]
```

TECHNICAL NOTES:

1. Object coding produced: When the

   SUBTRACT CORRESPONDING...

   option is used, for each subtraction and storage of items, coding is generated as though a SUBTRACT verb statement had been written for each; however, the SIZE ERROR coding generated is only for the last pair of operands subtracted.

   The use of the

   SUBTRACT CORRESPONDING

   option necessitates additional processing for compilation and can add appreciably to compilation time.

2. If the CORRESPONDING option is used, multiple operations are performed. The operations are defined by matching the data-names of numeric elementary items subordinate in hierarchy to data-name-1 and data-name-2. Data-names match if they and all their possible qualifiers up to, but not including, data-name-1 and data-name-2 are the same.

3. When the CORRESPONDING option is used, only the first complete description of any area is considered in the case where a REDEFINES has been used. All items describing the data contained within a REDEFINES group are ignored.

4. If the CORRESPONDING option is used, no items in the group referred to can contain an OCCURS clause.

5. In the CORRESPONDING option, data-name-1, and data-name-2 must not have a level number of 77 or 88.

6. For a discussion of the general rules regarding SUBTRACT, see Section VI.

APPENDIX A

COBOL SYSTEM RESERVED WORDS

This section contains four lists of words reserved for COBOL compilers and should be used by the programmer, depending upon the needs of the source program:

1. The first list contains all the words and symbols which are reserved for COBOL Compiler D. If the source program is to be compiled only on COBOL Compiler D, this is the only list needed to check the "legality" of programmer-supplied source program words. The programmer is reminded that, if upward-compatibility is a concern, the second list should be used.

2. The second list contains all the words and symbols which are reserved for COBOL Compiler H. If the source program is to be compiled only on COBOL Compiler H, this is the only list needed to check the validity of programmer-supplied source program words. The programmer is reminded that if upward-compatibility is a concern, the third list should be used.

3. The third list contains all the words (and all the symbols used in relations, formulas, and arithmetic statements) which are reserved for the Honeywell COBOL compilers. It is a combination of the Complete Word List in the DOD COBOL '65 Manual and the fixed names (of hardware units, input/output techniques, etc.) used in the Honeywell compilers. No programmer-supplied words in a source program (procedure-name, data-name, etc.) can be the same as any of these reserved words.

   This list should be consulted when any doubt exists as to the legality of any programmer-supplied word in a source program to be compiled by a Honeywell COBOL compiler. To insure compatibility, the programmer should always follow this reserved words list.

4. The fourth list is an abstract from the second list and contains those words in the Honeywell set of reserved words that are not included in the complete DOD list of reserved words.

   When a source program written for compilation on another COBOL compiler is to be compiled by a Honeywell COBOL compiler, and it is known that no programmer-supplied word of the source program duplicates a word in the DOD list, the first or fourth list (depending on the compiler to be used) should be used to check programmer-supplied words to be sure no Honeywell reserved word is duplicated.

COBOL D RESERVED WORDS

=
)
(
*BLOCK-COUNT
*BLOCK-COUNT-120
*CURRENT-DATE
*CHECK-POINT
*DATE-WRITTEN
*DATE-WRITTEN-120
*DIRECTION
*EXCESS-FILLER
*FILE-NO.
*FILLERA-120
*FILLERB-120
*FILLERC-120
*HALT-NAME
*IDENTIFICATION
*IDENTIFICATION-120
*LABEL-EXCESS
*MODE
*POSITION
*PROGRAM
*PURGE-DATE
*PURGE-DATE-120
*RECORD-COUNT
*REEL-NUMBER
*REEL-NUMBER-120
*SCOPE-PARAMS-1
*SCOPE-PARAMS-2
*SEGMENT
*SENTINEL
*STANDARD-LABEL
*STANDARD-LABEL-120

COBOL D RESERVED WORDS

*TAPE-NO.
*VISIBILITY

ABOUT
ACCEPT
ACCEPT-CARD-READER
ACCEPT-CONSOLE
ADD
ADDRESS
ADVANCING
AFTER
ALL
ALPHABETIC
ALPHANUMERIC
ALTER
ALTERNATE
AN
AND
APPLY
ARE
AREA
AREAS
ASSIGN
AT
AUTHOR
AUX-CHANNEL

BCD
BEFORE
BEGINNING
BEGIN/PROG/AT
BLANK
BLOCK
BY

CALL
CARD-PUNCH
CARD-READER
CARD-READERS
CHANNEL
CHANNELS
CHARACTER
CHARACTERS
CHECK
CLASS
CLOCK-UNITS
CLOSE
COBOL
COMMON-W-STORAGE
COMP
COMP-1
COMPUTATIONAL
COMPUTATIONAL-1
COMPUTER
CONFIGURATION
CONSTANT
CONTAINS
CONTINUE PROG AT
CONTROL
COPY
CORRESPONDING

DATA
DATE-COMPILED
DATE-WRITTEN
DECLARATIVES
DEPENDING
DIGIT
DIGITS
DISPLAY
DISPLAY-1
DISPLAY-2
DISPLAY-CONSOLE
DISPLAY-PRINTER
DIVIDE
DIVISION
DOLLAR

ELSE
END
ENDING
ENVIRONMENT
EQUAL
EQUALS
ERROR
EVERY
EVF-SIGNAL
EXAMINE
EXCEEDS
EXIT

FD
FILE
FILE-CONTROL
FILLER
FIRST
FLOAT
FOR
FROM

GIVING
GO
GREATER

HIGH-VALUE
HIGH-VALUES
H-200
H-200-SPECIAL
HLT-CTL
HONEYWELL-200
HVF-SIGNAL

ID
IDENTIFICATION
IF
I-O-CONTROL
IN
INCLUDE
INPUT
INPUT-OUTPUT
INSTALLATION
INTO
IS

JUSTIFIED

LABEL
LEADING
LEAVING
LEFT
LESS
LIBRARY
LINE
LINES
LOAD
LOCATION
LOCK
LOWER-BOUND
LOWER-BOUNDS
LOW-VALUE
LOW-VALUES

MEMORY
MODE
MODULE
MODULES
MOVE
MULTIPLE
MULTIPLY

NEGATIVE
NEXT
NO
NOT
NOTE
NUMERIC

OBJECT-COMPUTER
OBJECT-PROGRAM
OCCURS
OF
OFF
OMITTED
ON
OPEN
OPTIONAL
OR
OTHERWISE
OUTPUT

PAGE

COBOL D RESERVED WORDS (cont)

PERFORM
PICTURE
PICTURE@
PLACE
PLACES
POINT
POSITIVE
PRINTER
PRINTERS
PROCEDURE
PROCEED
PROGRAM-ID
PROTECT

QUOTE
QUOTES

RANGE
READ
READER-PUNCH
READER-PUNCHES
REASSIGNMENT
RECORD
RECORDING
RECORDS
REDEFINES
REEL
REMARKS
RENAMES
RENAMING
REPLACING
RERUN
RESERVE
REWIND
RIGHT
ROUNDED
RUN
RWCS

SAME
SECTION
SECURITY
SEGMENTATION
SEGMENT-LIMIT
SELECT
SENSE-SWITCH
SENTENCE
SEQUENCED
SHORT-GAP
SIGN
SIGNED
SIZE
SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
STANDARD-80
STANDARD-120
STATUS
STOP
SUBTRACT
SUPERVISOR
SUPPRESS
SYNCHRONIZED
TALLYING
TAPE
TAPE-UNIT
TAPE-UNITS
THAN
THEN
THROUGH
THRU
TIMES
TO
TRAILING-COUNT

UNEQUAL
UNTIL
UPON
UPPER-BOUND
UPPER-BOUNDS
USAGE
USE
USING

VALUE
VALUES
VARYING

WHEN
WITH
WORKING-STORAGE
WRITE

ZERO
ZEROES
ZEROS

COBOL H RESERVED WORDS

*BLOCK-COUNT
*BLOCK-COUNT-120
*CURRENT-DATE
*CHECK-POINT
*DATE-WRITTEN
*DATE-WRITTEN-120
*DIRECTION
*EXCESS-FILLER

*FILE-NO.
*FILLERA-120
*FILLERB-120
*FILLERC-120
*HALT-NAME
*IDENTIFICATION
*IDENTIFICATION-120
*LABEL-EXCESS
*MODE
*POSITION
*PROGRAM
*PURGE-DATE
*PURGE-DATE-120
*RECORD-COUNT
*REEL-NUMBER
*REEL-NUMBER-120
*SCOPE-PARAMS-1
*SCOPE-PARAMS-2
*SEGMENT
*SENTINEL
*STANDARD-LABEL
*STANDARD-LABEL-120
*TAPE-NO.
*VISIBILITY

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

COBOL H RESERVED WORDS (cont)

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
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

COBOL H RESERVED WORDS (cont)

LOWER-BOUND
LOWER-BOUNDS
LOW-VALUE
LOW-VALUES

MEMORY
MEMORY-DUMP
MEMORY-DUMP-KEY
MINUS

PREPARED
PRINT
PRINTER
PRINTERS
PRINT-TAPE
PRIORITY
PROCEDURE
PROCEED

SELECTED
SENSE-SWITCH
SENTENCE
SENTINEL
SEQUENCED
SET
SHORT-GAP
SIGN

COBOL H RESERVED WORDS (cont)

VALUES
VARYING

WHEN
WITH
WORDS
WORKING-STORAGE
WRITE

ZERO
ZEROES
ZERO

SPECIAL SYMBOLS

=
)
(
+
-
*
**
/

HONEYWELL COBOL RESERVED WORDS

=
+
-
*
/
**
(
)
.
*BLOCK-COUNT
*BLOCK-COUNT-120
*CURRENT-DATE
*CHECK-POINT
*DATE-WRITTEN
*DATE-WRITTEN-120
*DIRECTION
*EXCESS-FILLER
*FILE-NO.
*FILLERA-120
*FILLERB-120
*FILLERC-120
*HALT-NAME
*IDENTIFICATION
*IDENTIFICATION-120
*LABEL-EXCESS
*MODE
*POSITION
*PROGRAM
*PURGE-DATE
*PURGE-DATE-120
*RECORD-COUNT
*REEL-NUMBER
*REEL-NUMBER-120
*SCOPE-PARAMS-1
*SCOPE-PARAMS-2
*SEGMENT
*SENTINEL
*STANDARD-LABEL
*STANDARD-LABEL-120
*TAPE-NO.
*VISIBILITY

ABOUT
ACCEPT
ACCEPT-CONSOLE
ACCESS
ACTUAL
ADD
ADDRESS
ADVANCING
AFTER
ALL
ALPHABETIC
ALPHANUMERIC
ALTER
ALTERNATE
AN
AND
APPLY
ARE
AREA
AREAS
AS
ASCENDING
ASSIGN
AT
AUTHOR
AUX-CHANNEL

BANK
BANKS
BATCH-OPTION
BEFORE
BEGIN/PROG/AT
BEGINNING
BEGINNING-FILE-LABEL
BEGINNING-TAPE-LABEL
BIT
BITS
BLANK
BLOCK
BLOCK-COUNT
BLOCK-SIZE
BREAKPOINT-n
BY

CAL-400
CAL-800
CAL-1800
CARD-PUNCH
CARD-PUNCH-INTERLOCKED
CARD-READ
CARD-READ-INTERLOCKED
CARD-READER
CARD-READER-xx
CARD-READERS
CARD STORAGE

HONEYWELL COBOL RESERVED WORDS (cont)

CARD-TAPE-READ
CF
CH
CHANNEL-x
CHANNELS
CHARACTER
CHARACTERS
CHECK
CLASS
CLOCK-UNITS
CLOSE
COBOL
CODE
COLUMN
COMMA
COMMERCIAL-COLLATING-SEQUENCE
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
CONVERSION
COPY
COPY-READ-WRITE
CORRESPONDING
COUNT
CURRENCY

DATA
DATE-COMPILED
DATE-WRITTEN
DE
DECIMAL POINT
DECLARATIVES
DEFINE
DEMAND-READ
DEMAND-WRITE
DEPENDING
DESCENDING
DETAIL
DIGIT
DIGITS
DISPLAY
DISPLAY-CONSOLE
DISPLAY-PRINTER
DIVIDE
DIVIDED
DIVISION
DOLLAR
DOWN
DUMP
DUMP-FILE

ELSE
END
END-OF-FILE
END-OF-TAPE
ENDING
ENDING-FILE-LABEL
ENDING-TAPE-LABEL
ENTER
ENVIRONMENT
EQUAL
EQUALS
ERROR
EVERY
EVF-SIGNAL
EXAMINE
EXCEEDS
EXIT
EXPONENTIATED

FD
FILE
FILE-CONTROL
FILE-LABEL
FILE-LIMIT
FILE-LIMITS
FILE-NO
FILLER
FILLING
FINAL
FIRST
FIXED-LENGTH
FLOAT
FLOATING-POINT
FOOTING
FOR
FORMAT
FROM

GENERATE
GIVING
GO
GREATER
GROUP

H-n
H-200
H-200-SPECIAL
H-400
H-800
H-1400
H-1800
HASHED
HEADING
HIGH-VALUE
HIGH-VALUES
HOLD
HONEYWELL-n
HONEYWELL-200
HONEYWELL-200-SPECIAL
HONEYWELL-400
HONEYWELL-800
HONEYWELL-1400
HONEYWELL-1800
HVF-SIGNAL

I-O
I-O-CONTROL
ID
IDENTIFICATION
IF
IN
INCLUDE
INDEX
INDEXED
INDICATE
INITIATE
INPUT
INPUT-OUTPUT
INSTALLATION
INTO
INVALID
IS

JUSTIFIED

KEY
KEYS
KEY-SIZE

LABEL
LABEL-EXCESS
LAST
LEADING
LEAVING
LEFT
LESS
LIBRARY
LIMIT
LIMITS
LINE
LINE-COUNTER
LINES
LOCATION

HONEYWELL COBOL RESERVED WORDS (cont)

LOCK
LOW-VALUE
LOW-VALUES
LOWER-BOUND
LOWER-BOUNDS

MEMORY
MEMORY-DUMP
MEMORY-DUMP-KEY
MERGE-READ-WRITE
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
OH
OMITTED
ON
OPEN
OPTIONAL
OR
OTHERWISE
OUTPUT
OV
OVERFLOW
OVERLAP-READ
OVERLAP-WRITE

PAGE
PAGE-COUNTER
PAPER-TAPE-PUNCH
PAPER-TAPE-READER
PERFORM
PF
PH
PICTURE
PLACE
PLACES
PLUS
POINT
POSITION
POSITIVE
PREPARED
PRINT
PRINT-STORAGE
PRINT-TAPE
PRINTER
PRINTER-xx
PRINTERS
PRIORITY
PROCEDURE
PROCEDURE/DIVISION
PROCEED
PROCESSING
PROGRAM-ID
PROTECT
PROTECTION
PUNCH
PUNCH-xx
PUNCH-TAPE
PUNCHES
PURGE-DATE

QUOTE
QUOTES

RANGE
RD
READ
READER-PUNCH
READER-PUNCHES
RECORD
RECORD-COUNT
RECORDING
RECORD-SIZE
RECORDS
REDEFINES
REEL
REEL-NUMBER
RELEASE
REMARKS
RENAMES
RENAMING
REPLACING
REPORT
REPORTING
REPORTS
RERUN
RESERVE
RESET
RETURN
REVERSED
REWIND
RF
RH
RIGHT
ROUNDED
RUN

SA
SAME
SCOPE-PARAMS-1
SCOPE-PARAMS-2
SD
SEARCH
SECTION
SECURITY
SEEK
SEGMENTATION
SEGMENT-LIMIT
SELECT
SELECTED
SENSE-SWITCH
SENTENCE
SENTINEL
SEQUENCED
SEQUENTIAL
SET
SIGN
SIGNED
SIZE
SLAVE-TYPEWRITER
SORT
SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
STANDARD-LABEL
STATUS
STOP
SUBTRACT
SUM
SUPERVISOR
SUPPRESS
SYMBOLIC
SWITCH-1
SWITCH-2
SWITCH-3
SWITCH-4
SYNCHRONIZED

TALLY
TALLYING
TAPE
TAPE-m
TAPE-NO

HONEYWELL COBOL RESERVED WORDS (cont)

TAPE-READ
TAPE-READ-BACKWARD
TAPE-READ-FORWARD
TAPE-UNIT
TAPE-UNIT-xx
TAPE-UNITS
TAPE-WRITE
TEMP-STORAGE
TERMINATE
TEST-PATTERN
THAN
THEN
THROUGH
THRU
TIMES
TO
TODAYS-DATE
TRAILING-COUNT
TYPE

UNEQUAL
UNIT
UNTIL
UP
UPON
UPPER-BOUND
UPPER-BOUNDS
USAGE
USE
USING

VALUE
VALUES
VARIABLE-LENGTH
VARYING

WHEN
WITH
WORD
WORDS
WORKING-STORAGE
WRITE

ZERO
ZEROES
ZEROS

HONEYWELL COBOL RESERVED WORDS NOT PART OF DOD RESERVED WORDS

=
(
)
.
*BLOCK-COUNT
*BLOCK-COUNT-120
*CURRENT-DATE
*CHECK-POINT
*DATE-WRITTEN
*DATE-WRITTEN-120
*DIRECTION
*EXCESS-FILLER
*FILE-NO.
*FILLERA-120
*FILLERB-120
*FILLERC-120
*HALT-NAME
*IDENTIFICATION
*IDENTIFICATION-120
*LABEL-EXCESS
*MODE
*POSITION
*PROGRAM
*PURGE-DATE
*PURGE-DATE-120
*RECORD-COUNT
*REEL-NUMBER
*REEL-NUMBER-120
*SCOPE-PARAMS-1
*SCOPE-PARAMS-2
*SEGMENT
*SENTINEL
*STANDARD-LABEL
*STANDARD-LABEL-120
*TAPE-NO.
*VISIBILITY

ACCEPT-CARD-READER
ACCEPT-CONSOLE
AUX-CHANNEL

BEGIN/PROG/AT
BLOCK-SIZE

CARD-PUNCH
CARD-READ
CARD-READER
CARD-READERS
CARD-TAPE-READ
CHANNEL
COMMERCIAL-COLLATING-SEQUENCE
COMP
COMP-n
COMPUTATIONAL-n
COMPUTER
CONSOLE-KEYBOARD
CONSOLE-TYPEWRITER
CONTINUE/PROG/AT

DISPLAY/CONSOLE
DISPLAY/PRINTER
DUMP
DUMP-FILE
FILE-LABEL
FILE-NO
FLOATING-POINT

H-200-SPECIAL
HONEYWELL-nnn(n)

KEY-SIZE

LABEL-EXCESS

MODULE
MULTIPLY-DIVIDE

PRINT
PRINT-STORAGE
PRINT-TAPE
PRINTER
PRINTER-nn
PRINTERS
PUNCH
PUNCH-TAPE
PAPER-TAPE-PUNCH
PAPER-TAPE-READER
PUNCHES

READER-PUNCH
READER-PUNCHES
RECORD-SIZE

SCOPE-PARAMS-1
SCOPE-PARAMS-2
SEGMENTATION
SENSE-SWITCH

HONEYWELL COBOL RESERVED WORDS NOT PART OF DOD RESERVED WORDS (cont)

SLAVE-TYPEWRITER
STANDARD-LABEL
TAPE-NO
TAPE-READ
TAPE-READ-BACKWARDS
TAPE-READ-FORWARD
TAPE-UNIT
TAPE-UNITS
TAPE-WRITE
TEMP-STORAGE
TRAILING-COUNT

APPENDIX C
PROGRAMMING TIPS

This section contains additional information useful to the programmer. Programming tips will be added as they attain significant importance as defined by Honeywell.

REFERENCING TAPE LOADER-MONITOR C

General

To allow the user flexibility when running a string of object programs, seven fields can be referenced in Tape Loader-Monitor C. These fields would normally be used to determine the program to be loaded when the current program exists. Each field is assigned a reserved word and has an implied CLASS and SIZE. Normal Series 200 COBOL rules apply when referencing or changing the values in these fields.

The loading of non-resident segments will not affect these fields. (The fields will be saved and restored by the segment control routine.) Conversly, modification of these fields by the source program will not affect segment loading.

Parameters

Program Name:

1. Name - *PROGRAM (The * is part of the name.)
2. CLASS - Alphanumeric.
3. SIZE - 6
4. Value at load time - External program name.
5. Location (Octal) - 104 through 111.

Segment Name:

1. Name - *SEGMENT
2. CLASS - Alphanumeric.
3. SIZE - 2
4. Value at load time - 00
5. Location (Octal) - 112 through 113.

Search Direction:

1. Name - *DIRECTION

2. CLASS - Alphanumeric.
3. SIZE - 1
4. Value at load time - B
5. Location (Octal) - 152
6. Values
   a. B - Search forward.
   b. C - Search backward.

Relative position:

1. Name - *POSITION
2. CLASS - alphanumeric.
3. SIZE - 1
4. Value at load time - 1
5. Location (Octal) - 156
6. Values - This contains an octal number to be used by the loader when it is searching by visibility and relative position (see search mode, below).

Note - The value of this field must be octal and any references to it must be alphanumeric. Thus, the user must be careful when dealing with values greater than 8.

The loader recognizes each segment as a separate loading unit. Relative position 0 is undefined. The last unit is reloaded by setting relative position to 1 and direction to backwards. The next unit is loaded by setting relative position to 1 and direction to forward.

Search Mode:

1. Name - *MODE
2. CLASS - alphanumeric
3. SIZE - 1
4. Value at load time - The value is the same as was set by the previous program in the string. If the current program is the first in the string, the value will be 20 (octal) if loaded by name and 01 (octal) if loaded by visibility.
5. Location (octal) - 157.
6. Values.
   a. 01 (octal) search by visibility and relative position - Load

c. 00 (octal) search for and load the unit with the specified segment name within the current program. If the loader reads a segment with a program name different than the program name in 104 through 111, the loader will halt.

d. 60 (octal) search for and load the unit with specified pro-

Current Date:

1. Name - *CURRENT-DATE
2. CLASS - alphanumeric
3. SIZE - 5
4. Location (octal) - 216 through 222
5. Value - The Julian date taken from the equipment card.

eg. 127th day of 1967 = 12767

PAPER TAPE READ/PUNCH CAPABILITY

General

Paper tape read/punch devices may replace card read/punch devices on Honeywell Series 200 COBOL Compilers D and H. Certain initialization procedures must be performed. These are discussed in the following paragraphs.

Initialization

It is necessary for the user to reassemble a number of programs in order to fit the paper tape requirements of his particular installation. These programs are:

COBOL Compiler D	COBOL Compiler H
COBOLD	COBOLH
ABABEG	ABBBEG
ABARN1	ABBRN1
ABARN2	ABBRN2
ABAOBJ	ABBOBJ
ABA19W	ABB19W
ABAP13	ABBP13
ABAOBG	ABBOBG

A translation table must be entered into each program listed above. This table is used to translate paper tape code into internal code for input and vice versa for output. Information regarding line numbers, tags, etc., is provided as necessary by Honeywell.

Input Translation Table

When decoding paper tape, it is conceivable and in certain cases mandatory (with 5-channel tape) that one paper-tape character should denote two different internal characters. This is accomplished by using two translation tables. References to n-channel tape include parity within the count n; i.e., 7-channel tape with even parity indicates 6 data bits and one bit for parity.

If one translation table is used, it must be assembled into the various runs starting at the tag TABLEA.

If two tables are needed, the first must be assembled into the runs at tag TABLEA and the second at tag TABLEB. The various options are as follows:

Number of Channels Punched	Number of Tables Allowed
5	must have two tables to accommodate all necessary internal characters - 32 characters in each table
6	one or two tables - 64 characters in each table
7	one or two tables - 128 characters in each table
8	one table only - 256-character table

Each entry in these tables comprises one memory location. The order of characters in the table is determined by their corresponding paper-tape characters. For example, let us suppose that we are reading tape which has, at most, 7 characters punched in any one frame (six data bits plus parity or seven data bits). Consider the paper-tape character:

$$0111010 = 58_{10}$$

If we wish this character to correspond to an A in core (21_8), then 21 must be the 59th entry in the table. The possible tape characters for 7-channel tape are:

$$000_8 - 177_8 = 0_{10} - 127_{10}$$

Thus, the table for 7-channel tape must contain 128 entries. No punctuation is necessary within the input tables.

Output Translation Table

The output translation table is always 128 characters in length. It is comprised of 64 two-character entries. When an item mark is found in the high-order location of an entry, a change table character is issued prior to punching that character. The 64 entries correspond to the 64 internal characters; i.e., the first entry represents the internal character 00 (in bits). The 22nd entry represents an A on paper tape (in bits).

When punching n channels, only the low order n bits are used from each entry in the table. For example, in order to punch channels 7, 5, 4, 3, and 1 into 8-channel tape and to have this character represent the letter N, the 46th entry in the table must contain:

000	001	011	101
0	1	3	5

Card Image Preparation

Through the use of special characters, card images may be compacted, thus eliminating unused card columns. Each of the following functions should have special characters (tape and internal) assigned to it by the user. This character must not be used for any other purpose.

Ignore	Delete Line
End Field	End Reel
End Line	Change Table

It is recommended that several "ignore" characters be inserted between control cards to improve readability.

Operating Instructions

The following sequence of instructions apply to compile-time processing:

1. A halt occurs at 17002
2. Key into location 100_8: 01
3. Key into location 104_8: $\begin{Bmatrix} \text{COBOLD010} \\ \text{COBOLH010} \end{Bmatrix}$
4. Key into location 227_8: 43_8
5. Key into location 230_8: trunk assignment for paper tape reader; either of the following entries – 4X (Parity not checked) 6X (Parity checked)
6. Key into location 231_8: parameter from the following chart

	Column	Content	Meaning
Input	27	L	
	28	4X or 6X	Trunk assignment
	29	zz	zz = parameter from above chart
Output	33	L	
	34	0X	Trunk assignment
	35	zz	zz = parameter from the following chart

Maximum No. of Bits to be Punched Per Frame	Output Parameter
5	40
6	40
7	00
8	00

2. Console Call card
3. COBOL*INPUT card
4. OPTION card
5. COBOL source program

Refer to the publication Paper Tape Equipment, DSI-322, for further information on paper tape capability.

SAVING MEMORY

1. Procedure Division Segmentation. A program can generally be segmented, especially in the opening (or housekeeping) and closing (or ending) sections. A nonresident segment is automatically called by a GO TO or PERFORM statement referencing a procedure-name (section- or paragraph-name) within that segment. The call searches the BRT for the segment and loads it into the area of memory above that specified for the resident segments. This segment loads in over another nonresident segment and does not preserve the setting of any alterable GO TO switches (set by the user or by a PERFORM). For this reason a nonresident segment cannot be PERFORMed and then a PERFORM or GO TO another nonresident segment from within

3. SAME AREA - Purpose of the SAME AREA option in the I/O CONTROL

LIMITATIONS WITHIN THE COBOL D COMPILER

1. 63 OCCURS statements per program.
2. 4096 characters per record.

   NOTE: All 01 levels are records to the logic of the compiler, even in Working-Storage.
3. 31 CORRESPONDING statements per program.
4. 30 characters per word (i.e. data-name or procedure-name).
5. 30 88 levels per program.
6. 35 identical data-names per program.
7. 4 MULTIPLE-FILE clauses per program.
8. 15 files per program.

   NOTE: Working-Storage and Constant Sections are each considered a file.
9. MOVE CORRESPONDING with either sending or receiving group containing more than 85 data-names does not work correctly. This refers to all data-names within the group, not just those which correspond.