Burroughs

B 2500
and
B 3500
SYSTEMS

COBOL REFERENCE MANUAL

Burroughs

B 2500/B 3500

INFORMATION PROCESSING SYSTEM

COBOL

REFERENCE MANUAL

Burroughs Corporation
Detroit, Michigan 48232

$5.00

Printed in U.S. America

5-69

1033099

Burroughs Corporation believes the program described in this manual to be accurate and reliable, and much care has been taken in its preparation. However, the Corporation cannot accept any responsibility, financial or otherwise, for any consequences arising out of the use of this material. The information contained herein is subject to change. Revisions may be issued to advise of such changes and/or additions.

This reprint includes the information released under the following:

PCN 1033099-001 (Aug 28, 1969)
PCN 1033099-002 (Oct 8, 1969)
PCN 1033099-003 (Nov 11, 1969)

Correspondence regarding this document should be forwarded using the Remarks Form at the back of the manual, or may be addressed directly to Systems Documentation, Sales Technical Services, Burroughs Corporation, 6071 Second Avenue, Detroit, Michigan 48232.

ACKNOWLEDGEMENT

Any organization interested in reproducing the COBOL report and specifications in whole or in part, using ideas taken from this report as the basis for an instruction manual or for 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.

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

No warranty, expressed or implied, is made by any contributor or by the COBOL Committee as to the accuracy and functioning of the programing system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection therewith.

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

The authors and copyright holders of the copyrighted material used herein

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

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 programing manuals or similar publications.

TABLE OF CONTENTS

TABLE OF CONTENTS (cont)

TABLE OF CONTENTS (cont)

TABLE OF CONTENTS (cont)

TABLE OF CONTENTS (cont)

TABLE OF CONTENTS (cont)

TABLE OF CONTENTS (cont)

TABLE OF CONTENTS (cont)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (cont)

LIST OF TABLES

LIST OF TABLES (cont)

INTRODUCTION

This manual provides a complete description of COBOL (COMMON BUSINESS ORIENTED LANGUAGE) as implemented for use on the Burroughs B 2500/B 3500 Electronic Data Processing Systems. This concept of COBOL embraces the adoption of proposed American National Standards Institute (ANSI) COBOL-68.

COBOL's long list of advantages is derived chiefly from its intrinsic quality of permitting the programmer to state the problem solution in English. The programing language reads much like ordinary English prose, and can provide automatic program and system documentation. When users adopt in-house standardization of elements within files, plus well chosen data-names, before attempting to program a system, they obtain maximum documentational advantages of the language described herein.

To a computer user, the Burroughs B 2500/B 3500 COBOL offers the following major advantages:

a. Expeditious means of program implementation.

b. Accelerated programmer training and simplified retraining requirements.

c. Reduced conversion costs when changing from a computer of one manufacturer to that of another.

d. Significant ease of program modification.

e. Standardized documentation.

f. Documentation which facilitates non-technical management participation in data processing activities.

g. Efficient object program code.

h. Segmentation capability which sets the maximum allowable program size well in excess of any practical requirement.

i. Due to the incorporation of debugging language statements, a high degree of sophistication in program design is achieved.

j. A comprehensive source program diagnostic capability.

A program written in COBOL, called a source program, is accepted as input by the B 2500/B 3500 COBOL Compiler. The compiler verifies that all rules outlined in this manual are satisfied, and translates the source program language into an object program language capable of communicating with the computer and directing it to operate on the desired data. Should source corrections become necessary, appropriate changes can be made and the program recompiled. Thus, the source deck always reflects the object program being operationally executed.

A COBOL source program is always divided into four parts or DIVISIONS in the following order:

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

The purpose of the IDENTIFICATION DIVISION is to identify the program and to include an overall description of the program.

The ENVIRONMENT DIVISION consists of two sections. The Configuration Section specifies the equipment being used. The Input-Output Section associates files with the hardware devices that will be used for their operation. This section also furnishes the compiler with information about mass storage parameters.

The DATA DIVISION is used to describe data elements which the object program is to manipulate or create. These data elements may be items within files, records or program work areas, and constants.

The PROCEDURE DIVISION defines the necessary steps which will accomplish the desired task when operating on the data as defined in the DATA DIVISION.

This publication supercedes and replaces the Burroughs B 2500/ B 3500 COBOL Language Manual Form 1027406 dated June, 1968.

SECTION 1

COBOL LANGUAGE ELEMENTS

GENERAL.

It has been stated that COBOL is a language based on English and that the language is composed of words, statements, sentences, paragraphs, etc. The following paragraphs define the rules to be followed in the creation of this language. The use of the different constructs formed from the created words is covered in subsequent sections of this document.

CHARACTER SET.

The B 2500/B 3500 COBOL character set consists of the following 53 characters:

0 - 9	
A - Z	
	space or blank
+	plus sign
-	minus sign or hyphen
*	asterisk
/	slash (virgule)
=	equal sign
$	dollar sign
,	comma
.	period or decimal point
;	semicolon
"	quotation mark
(	left parenthesis
)	right parenthesis
>	greater than symbol
<	less than symbol
:	colon
@	at sign

CHARACTERS USED FOR WORDS.

The character set for words consists of the following 37 characters:

0 - 9

A - Z

- (hyphen)

PUNCTUATION CHARACTERS.

The following characters may be used for punctuation:

@	at sign		space or blank
"	quotation mark	.	period
(	left parenthesis	,	comma (see note below)
)	right parenthesis	;	semicolon (see note below)

NOTE

Commas and semicolons may be used between statements, at the programmer's discretion, for enhanced readability of the source program. Use of these characters implies that a following statement is to be included as a portion of an entire statement.

CHARACTERS USED IN EDITING.

The COBOL Compiler accepts the following characters in editing:

$	dollar sign	+	plus
*	asterisk (check protect)	-	minus
,	comma	CR	credit
.	actual decimal point	DB	debit
B	space	Z	zero suppress
0	zero		

CHARACTERS USED IN FORMULAS.

The COBOL Compiler accepts the following characters in arithmetic expressions:

+	addition	**	exponentiation
-	subtraction	(	left parenthesis
*	multiplication	)	right parenthesis
/	division		

CHARACTERS USED IN RELATIONS.

The COBOL Compiler accepts the following characters in conditional relations:

=	equal
<	less than
>	greater than

DEFINITIONS OF WORDS.

A word is created from a combination of not more than 30 characters, selected from the following:

A through Z
0 through 9
- (the hyphen)

A word is ended by a space, or by a period, comma, or semicolon. A word may not begin or end with a hyphen. (A literal constitutes an exception to these rules, as explained later.)

TYPES OF WORDS.

COBOL (like English) contains types of words. These word types are:

a. Nouns.
b. Verbs.
c. Reserved words.

NOUNS.

Nouns are divided into nine special categories:

a. File-names.
b. Record-names.
c. Data-names.
d. Condition-names.
e. Procedure-names.
f. Literals.
g. Figurative constants.

h. Special register names.

i. Special names.

Since the noun is a word, its length may not exceed thirty characters (exception: literals may not exceed 160 characters). For purposes of readability, a noun may contain a hyphen. However, the hyphen may neither begin nor end the noun (this does not apply to literals).

FILE-NAME.. A file-name is a collective name or word assigned to designate a set of data items. The contents of a file are divided into logical records that in turn are made up of any consecutive set of data items.

RECORD-NAME. A record-name is a noun assigned to identify a logical record. A record can be sub-divided into several data items, each of which is distinguishable by a data-name.

DATA-NAME. A data-name is a noun assigned to identify elements within a record or work area and is used in COBOL to refer to an element of data, or to a defined data area containing data elements. Each data-name must be composed of at least one alphabetical character.

CONDITION-NAME. A condition-name is a special data-name which is assigned to a specific value within a set of values. For illustrating a condition-name, consider this example. If THIS-YEAR identifies the twelve months of a year, whereas its subordinate data items are defined as JANUARY, FEBRUARY, etc., and the values assigned to each month range from 01 to 12, then it follows that JUNE would have the assigned value of 06. Using the condition-name JUNE, the programmer can utilize it in conditional statements as follows:

```
IF JUNE GO TO . . . .
```

which is logically equivalent to the statement:

```
IF THIS-YEAR IS EQUAL TO 06 GO TO . . . .
```

As a conditional-name, the special data-name itself is called a conditional-variable. The value that it may assume is referred to by condition-names. The condition-name is formatted according to noun rules and may be used only in conditional statements.

PROCEDURE-NAME. A procedure-name is either a paragraph-name or section name, and is formulated according to noun rules. The exception is that a procedure-name may be composed entirely of numeric characters. Two procedure-names are identical only if they both consist of the same character strings. For example: procedure-names 007 and 7 are not equivalent.

LITERALS. A literal is an item of data which contains a value identical to the characters being described. There are three classes of a literal: numeric, non-numeric, and undigit.

Numeric Literal.

A numeric literal is defined as an item composed of characters chosen from the digits 0 through 9, the plus sign (+) or minus sign (-), and the decimal point. The rules for the formation of a numeric literal are:

a. Only one sign character and/or more than one decimal point may be contained in a numeric literal for use with Sterling. Left-most decimal determines the scale.

NOTES

A comma must be substituted for the decimal point if the DECIMAL-POINT IS COMMA option is used (see SPECIAL-NAMES in the ENVIRONMENT DIVISION).

The implied USAGE of numeric literals is COMPUTATIONAL except when used with the verbs DISPLAY or STOP.

b. There must be at least one digit in a numeric literal.

c. The sign of a numeric literal must appear as the left-most character. If no sign is present, the literal is defined as a positive value.

d. The decimal point may appear anywhere within the literal except for the right-most character of a numeric literal. A decimal point within a numeric literal is treated as an implied decimal point. Absence of a decimal point denotes an integer quantity. (An integer is a numeric literal which contains no decimal point.)

e. A numeric literal used for arithmetic manipulations cannot exceed 99 signed digits, otherwise, the maximum is 160 digits. The following are examples of numeric literals.

```
 13247
  .005
+1.808
-.0968
7894.54
```

Non-Numeric Literal.

A non-numeric literal may be composed of any allowable character. The beginning and end of a non-numeric literal is denoted by a quotation mark. Any character enclosed within quotation marks is part of the non-numeric literal. Subsequently, all spaces enclosed within the quotation marks are considered part of the literal. Two consecutive quotation marks within a non-numeric literal cause a single quote to be inserted into the literal string. Four consecutive quotation marks will result in a single " literal.

A non-numeric literal cannot itself exceed 160 characters. Examples of non-numeric literals are:

Literal on source program level	Literal stored by compiler
"ACTUAL SALES FIGURE"	ACTUAL SALES FIGURE
"-1234.567"	-1234.567
"""LIMITATIONS"""	"LIMITATIONS"
"ANNUAL DUES"	ANNUAL DUES
""""	"
"A""B"	A"B

NOTE

Literals that are used for arithmetic computation must be expressed as numeric literals and must not be enclosed in quotation marks as non-numeric literals. For example, "-7.7" and -7.7 are not equivalent. The compiler stores the non-numeric literal as -7.7, whereas the numeric literal would be stored as $\bar{0}$077 if the PICTURE were S999V9 DISPLAY with the assumed decimal point located between the two sevens.

Undigit Literals.

Binary 10 through 15 are represented as A through F and must be bounded by @ signs. For example, binary 11 would be literalized by @B@. An undigit literal cannot exceed 160 digits. Refer to section 7 for the correct declaration.

FIGURATIVE CONSTANT. A figurative constant is a particular value that has been assigned a fixed data-name and must never be enclosed in quotation marks except when the word, rather than the value, is desired. The figurative constant names and their meanings are:

ZERO ZEROS ZEROES	Represents the value of 0.
SPACE SPACES	Represents one or more spaces (blanks).
HIGH-VALUE HIGH-VALUES	Represents the highest internal coding sequence (i.e., 999) value. When HIGH-VALUES

are moved to a signed numeric computational field, the sign will not be changed.

LOW-VALUE
LOW-VALUES

Represents the lowest internal coding sequence (blanks) value. When LOW-VALUES are moved to a signed numeric computational field, the sign will not be changed.

QUOTE
QUOTES

Represents one or more of the single character " (quotation mark). The word QUOTE or QUOTES does not have the same meaning in COBOL as the symbol ". For example, if "STANDARDS" appears as part of the COBOL source program, the word STANDARDS is stored in the object program. If however, the full "STANDARDS" is desired in a DISPLAY statement, it can be achieved by writing QUOTE "STANDARDS" QUOTE, in which case the object program will print "STANDARDS". The same result can be obtained by writing """STANDARDS""" in the source program. Only the latter method can be used in MOVE statements and conditionals.

ALL

When followed by a non-numeric literal or a figurative constant, the word ALL represents a series of that literal. For example, if the COBOL statement is MOVE ALL literal TO ERROR-CODE, then the resultant ERROR-CODE would take on the following values:

ALL literal	Size of ERROR-CODE	Resulting value of ERROR-CODE
ALL "ABC"	7 characters	ABCABCA
ALL "3" or ALL 3	5 characters	33333
ALL "HI-LO"	12 characters	HI-LOHI-LOHI
ALL QUOTE	3 characters	"""
ALL SPACES	9 characters	(nine spaces)

NOTE

The use of ALL with figurative constants, as illustrated in the last two instances, is redundant. MOVE ALL SPACES and MOVE SPACES would yield the same result.

SPECIAL REGISTER NAME. The Burroughs COBOL Compiler provides four special PROCEDURE DIVISION register names which are:

a. TALLY.
b. TODAYS-DATE (Calendar).
c. DATE (Julian).
d. TIME.

TALLY.

The special register TALLY is automatically provided by the COBOL Compiler and has a defined length of five COMPUTATIONAL digits. The primary use of TALLY is in conjunction with the EXAMINE statement, however, TALLY may be used as temporary storage or an accumulative area during the interim when EXAMINE...TALLYING...is not being executed in a program.

Todays-Date (Calendar).

This special register contains the current date and is maintained by the Master Control Program (MCP). Its format is made of three character pairs, each representing the month, day and year. For example, if the current date is Dec. 13th, 1968, the TODAYS-DATE register contains 121368. The function of TODAYS-DATE is to provide the programmer with a means of referring to the current date during program execution. TODAYS-DATE is maintained in COMPUTATIONAL form.

Date (Julian).

This special register contains the current Julian date and is maintained by the MCP. Its format is YYDDD. For example, if the current date were January 1, 1968, the DATE register would contain 68001. The function of DATE is to save programmatic evaluation of TODAYS-DATE when Julian dates are required. DATE is maintained in COMPUTATIONAL form.

Time.

Access to an internal clocking register reflecting the time of day is programmatically available whenever TIME is requested. This register is maintained in milliseconds by the MCP as a 10-digit COMPUTATIONAL field. The contents of the TIME register will be maintained in hours, minutes, seconds and 60th of seconds when TIME 60 is declared in the OBJECT-COMPUTER paragraph.

SPECIAL-NAMES.

The SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION allows the programmer to assign a significant character for a CURRENCY SIGN, and to declare DECIMAL-POINT as being a COMMA and to provide a means of relating implementor hardware-names to mnemonic-names as desired by the programmer.

VERBS.

Another type of COBOL word is a verb. A verb in COBOL is a single word that denotes action, such as ADD, WRITE, MOVE, etc. All allowable verbs in COBOL, with the exception of the word IF, are truly English verbs. The usage of the COBOL verbs takes place primarily within the PROCEDURE DIVISION.

RESERVED WORDS.

The third type of COBOL word is a reserved word. Reserved words have a specific function in the COBOL language and cannot be used out of context, or for any other purpose than the one for which they were intended. Reserved words are for syntactical purposes and can be divided into three categories:

a. Connectives.

b. Optional words.

c. Key words.

A complete list of reserved words in COBOL used by the compiler is included in Appendices A and E.

CONNECTIVES. Connectives are used to indicate the presence of a qualifier or to form compound conditional statements. The connectives OF and IN are used for qualification. On the other hand, AND, AND NOT, OR, or NOT are used as logical connectives in conditional statements.

OPTIONAL WORDS. Optional words are included in the COBOL language to improve the readability of the statement formats. These optional words may be included or omitted, as the programmer wishes. For example, IF A IS GREATER THAN B... is equivalent to IF A GREATER B..... Therefore, the inclusion or omission of the words IS and THAN does not influence the logic of the statement.

KEY WORDS. The third kind of reserved words is referred to as being a key word. The category of key words includes the verbs and required words needed to complete the meaning of statements and entries. The category also includes words that have a specific functional meaning. In the example shown in the above paragraph, the words IF and GREATER are key words.

STATEMENT AND SENTENCE FORMATION.

Statements are formed by the completion of the various entry and verb constructs discussed in the later sections of this manual. A statement may be terminated by a period and thus become a sentence. A group of statements, terminated by a period, forms a sentence. An example of a sentence made up of a group of statements would be MOVE A TO B, ADD 01 TO COUNTER WRITE SUMMARY. Note that the word THEN can be used interchangeably with the semi-colon or comma.

PARAGRAPH FORMATION.

One or more sentences may comprise a paragraph. A paragraph begins with a paragraph name and is terminated by the paragraph name of the next paragraph.

SECTION FORMATION.

One or more paragraphs may formulate a section. A section includes all paragraphs between one section name and a following section name or the end of the source program. The method of referring to procedures within sections and transferring of operational control to these procedures is discussed in the PROCEDURE DIVISION section.

NOTATION USED IN VERB AND ENTRY FORMATS.

The notation conventions that follow enable the reader to interpret the COBOL syntax presented in this manual.

KEY WORDS.

All underlined upper case words are key words and are required when the functions of which they are a part are utilized. Their omission will cause error conditions at compilation time. An example of key words is as follows:

$$\underline{\text{IF}}\quad \text{data-name}\quad \text{IS}\quad [\underline{\text{NOT}}]\quad \begin{Bmatrix}\underline{\text{NUMERIC}}\\ \underline{\text{ALPHABETIC}}\end{Bmatrix}$$

The key words are: IF, NOT, NUMERIC, and ALPHABETIC.

OPTIONAL WORDS.

All upper case words not underlined are optional words and are included for readability only and may be included or excluded in the source program. In the example above, the optional word is: IS.

LOWER CASE WORDS.

All lower case words represent generic terms which must be supplied in that format position by the programmer. Integer-1 and integer-2 are generic terms in the following example:

$$\underline{\text{FILE-LIMIT}}\ \text{IS integer-1}\ \underline{\text{THRU}}\ \text{integer-2}$$

BRACES.

When words or phrases are enclosed in braces { }, a choice of one of the entries must be made. In reference to the key words example above, one or the other of the words NUMERIC or ALPHABETIC must be included in the statement.

BRACKETS.

Words and phrases enclosed in brackets [ ] represent optional portions of a statement. If the programmer wishes to include the optional feature, he may do so by including the entry shown between brackets. Otherwise it may be omitted. In terms of the example above, the word enclosed in brackets is optional. However, if the programmer wishes to distinguish between NUMERIC and ALPHABETIC, he must choose one of the words enclosed in braces.

CONSECUTIVE PERIODS.

The presence of ellipsis (...) within any format indicates that the data immediately preceding the notation may be successively repeated, depending upon the requirements of problem solving.

PERIOD.

When a single period is shown in a format, it must appear in the same position whenever the source program calls for the use of that particular statement. A space after a period is not required, however, such a practice will enhance readability of the source program.

SECTION 2

IDENTIFICATION DIVISION

GENERAL.

The first part or division of the source program is the IDENTIFICATION DIVISION. Its function is to identify the source program and the resultant output of its compilation. In addition, the date the program was written, the date the compilation was accomplished, plus other pertinent information may be included in the IDENTIFICATION DIVISION.

The structure of this division is as follows:

```
[MONITOR...]

IDENTIFICATION DIVISION.

[PROGRAM-ID. Any COBOL word.]

[AUTHOR. Any entry.]

[INSTALLATION. Any entry.]

[DATE-WRITTEN. Any entry.]

[DATE-COMPILED. Any entry - replaced by the current date
                and time as maintained by the MCP.]

[SECURITY. Any entry.]

[REMARKS. Any entry. Continuation lines must be coded
          in Area B of the coding form.]
```

SYNTAX RULES.,

The following rules must be observed in the formation of the IDENTIFICATION DIVISION:

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

b. All paragraph-names within this division must begin under Area A of the coding form.

c. An entry following a paragraph-name cannot contain periods, except that one must be present to denote the end of that entry.

NOTES

When DATE-COMPILED is included, the compiler automatically inserts the time of compilation in the form of HH:MM and the date of compilation in the form of MM/DD/YY.

With the exception of the DATE-COMPILED paragraph, the entire division is copied from the input source program by the compiler and listed on the output listing for documentational purposes only.

MONITOR.

This statement provides a debugging trace of specified data-names.

Construct of this statement is:

```
[ MONITOR  [DEPENDING]  file-name   ( [data-name] ...   :

  [ { ALL                 } ] ).  ]
    { paragraph-name...   }
```

This statement must begin under Area A of the coding form. The parentheses and colon are required as part of the source program statement. MONITOR is active only while the file-name is in OPEN status.

Only one MONITOR statement per program is allowed and must precede the IDENTIFICATION DIVISION header card in the source program.

The file-name must be ASSIGNed to a line printer and is recognized by the compiler as being the output media for the MONITORed data-names.

The data-name(s) may be any name(s) appearing in the DATA DIVISION except for those which require subscripting or indexing.

Whenever a MONITORed elementary data-name is encountered as the receiving field in a MOVE or arithmetic statement, data-name and its current value are listed.

The MONITORing of paragraph-names within a USE AT END OF PAGE on the output file that is also the MONITOR file (e.g., same LINE PRINTER) will give undefined output when the physical End-of-Page condition is encountered.

If a group item appears in the data-name-list, it will be MONITORed only when explicitly used as a receiving field.

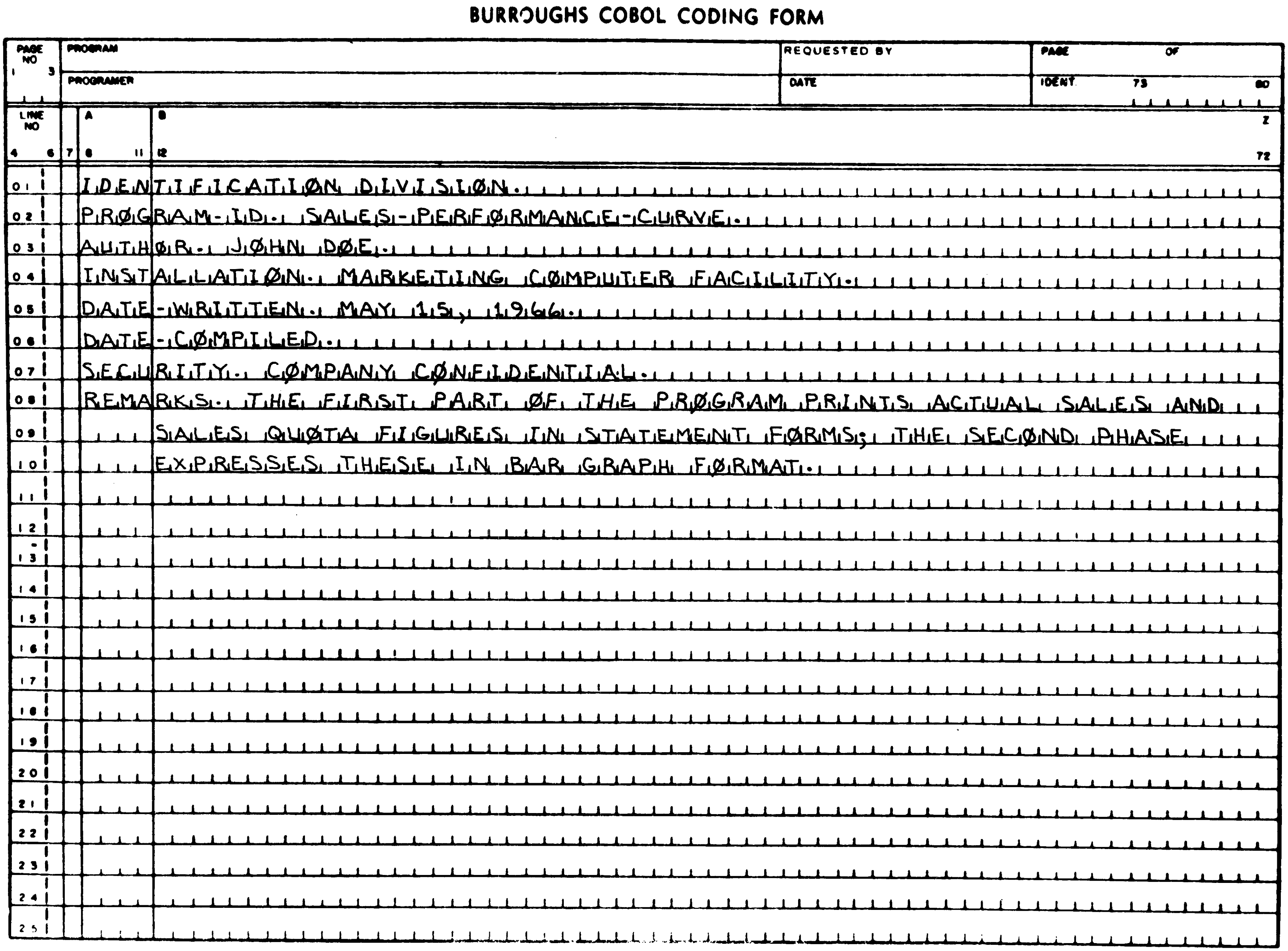
If the DEPENDING option is present, SW6 will be tested for an ON-OFF condition. Print of MONITORed items will depend upon the setting as being "ON".

All paragraph-names listed will be printed each time they are encountered, along with a total indicating the number of times that a paragraph-name has been passed. The total will be reset to zero whenever the paragraph-name, if in an overlayable segment, is overlayed in the operating program.

The use of the ALL option, instead of the paragraph-name list, will cause all section and paragraph-names to be MONITORed, thus providing a trace of the programs control path during operation.

CODING THE IDENTIFICATION DIVISION.

Figure 2-1 provides an illustrative example of how the IDENTIFICATION DIVISION may be coded in the source program. Note that continued lines must be indented to the B position of the form, or beyond.



BURROUGHS COBOL CODING FORM

PAGE NO 1 3	PROGRAM	REQUESTED BY	PAGE OF
	PROGRAMER	DATE	IDENT 73 80

```
LINE NO  A     B                                                           Z
4    6 7 8  11 12                                                          72
01       IDENTIFICATIØN DIVISIØN.
02       PRØGRAM-ID. SALES-PERFØRMANCE-CURVE.
03       AUTHØR. JØHN DØE.
04       INSTALLATIØN. MARKETING CØMPUTER FACILITY.
05       DATE-WRITTEN. MAY 15, 1966.
06       DATE-CØMPILED.
07       SECURITY. CØMPANY CØNFIDENTIAL.
08       REMARKS. THE FIRST PART ØF THE PRØGRAM PRINTS ACTUAL SALES AND
09           SALES QUØTA FIGURES IN STATEMENT FØRMS; THE SECØND PHASE
10           EXPRESSES THESE IN BAR GRAPH FØRMAT.
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
```

Figure 2-1. Coding the IDENTIFICATION DIVISION

SECTION 3

ENVIRONMENT DIVISION

GENERAL.

The ENVIRONMENT DIVISION is the second division of a COBOL source program. Its function is to specify the computer being used for the program compilation, to specify the computer to be used for object program execution, to associate files with the computer hardware devices, and to provide the compiler with pertinent information about disk storage files defined within the program. Furthermore, this division is also used to specify input-output areas to be utilized for each file declared in a program.

ORGANIZATION.

The ENVIRONMENT DIVISION consists of two sections. The CONFIGURATION SECTION contains the over-all specifications of the computer. The INPUT-OUTPUT SECTION deals with files to be used in the object program.

STRUCTURE.

The structure of this division is as follows:

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

SYNTAX RULES.

The following syntax rules must be observed in the formulation of the ENVIRONMENT DIVISION:

a. The ENVIRONMENT DIVISION must begin with the reserved words ENVIRONMENT DIVISION followed by a period.

b. All entries other than the ENVIRONMENT DIVISION source line are optional, but when used they must begin under Area A of the coding form.

Specific definitions for the ENVIRONMENT DIVISION paragraphs are given on the following pages.

CONFIGURATION SECTION.

The. CONFIGURATION SECTION contains information concerning the system to be used for program compilation (SOURCE-COMPUTER) and the system to be used for program execution (OBJECT-COMPUTER).

SOURCE-COMPUTER.

The function of this paragraph is to allow documentation of the configuration used to perform the COBOL compilation.

The construct of this paragraph is:

```
┌                       ┌ B-2500    ┐     ┐
│ SOURCE-COMPUTER.     <  B-3500     >  . │
└                       └ any entry ┘     ┘
```

This paragraph is for documentation only.

OBJECT-COMPUTER.

The function of this paragraph is to allow a description of the configuration used for the object program.

The construct of this paragraph is as follows:

```
[ OBJECT-COMPUTER.   [ { B-2500 } ]     [WITH SUPERVISOR CONTROL]
                     [ { B-3500 } ]

    MEMORY SIZE integer [ { WORDS      } ]      [TIME 60]
                        [ { CHARACTERS } ]
                        [ { MODULES    } ]

    [SEGMENT-LIMIT IS priority number]        ]
```

If section priority numbers are used in the PROCEDURE DIVISION, they must be positive integers with a value from zero through 99. The SEGMENT-LIMIT clause signifies the limit for non-overlayable program segmentation of sections numbered from 00 through 49. See SEGMENT CLASSIFICATION, PROGRAM SEGMENTS, AND PRIORITY NUMBERS on pages 5-19 through 5-23.

WITH SUPERVISOR CONTROL is documentational only and is ignored by the compiler.

The MEMORY SIZE clause is one of the factors used in determining the size of a COBOL object program and is normally only beneficial in programs containing the SORT verb or when a larger STACK is desired. If MEMORY SIZE is larger than the resultant COBOL object program, the recap at the end of the compilation will reflect the additional core in the size of the STACK.

The compiler will automatically determine MEMORY SIZE when one of the options of the clause is specified. The value of the integer will be multiplied times the specified option, giving a digit product, where:

CHARACTERS = 2 digits
WORDS = 4 digits
MODULES = 1000 digits

If a MEMORY SIZE option is not specified, the value of the integer entry will apply.

The MEMORY SIZE clause, if used, does not include core requirements for disk file headers. In addition to the core specified in the MEMORY SIZE clause, an additional 250 digits of memory are required for each disk file. This area is reserved in increments of 1000 digits and resides immediately behind the operating object program, just above the Program Limit Register, to avoid accidental destruction by internal operations being performed by the program.

The TIME 60 clause denotes that the contents of the internal TIME clocking register is to be maintained as hours, minutes, seconds and 60th of a second in the COMPUTATIONAL format: 00HHMMSS60, where 00 = zeros, HH = hours, MM = minutes, SS = seconds, 60 = 60th of a second.

SPECIAL-NAMES.

The function of this paragraph is to allow the programmer to assign a significant character for all currency signs, to declare decimal points as being commas and to provide a means of relating implementor hardware-names to user specified mnemonic-names.

The construct of this paragraph is:

```
[ SPECIAL-NAMES.    [CURRENCY SIGN IS literal]

 [implementor-name IS menmonic-name ...]

 [DECIMAL-POINT IS COMMA ].  ]
```

This paragraph is required if all decimal points are to be interchanged with commas and/or if all currency signs are to be represented by a character other than a dollar sign ($).

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

a. Numeric digits 0 through 9.

b. Alphabetic characters A, B, C, D, J, K, P, R, S, V, X, Z, or blank.

c. Special characters * + - , . ; ( ) ".

The clause DECIMAL-POINT IS COMMA signifies that the function of comma and period are to be exchanged in the PICTURE clause character-string and in numeric literals.

The implementor-name clause must be one of the allowable B 2500/ B 3500 COBOL hardware-names as listed on page 3-10. For example:

```
PUNCH IS CARD-PUNCH-EBCDIC
```

The mnemonic named device can be directly referred to in the ASSIGN clause.

The SPECIAL-NAMES paragraph statement ends with a period as a delimiter. Periods between clauses are not allowed.

INPUT - OUTPUT SECTION.

The INPUT-OUTPUT section contains information concerning files to be used by the object program.

FILE-CONTROL

The function of this paragraph is to name each file, to identify the file medium, and to specify a particular hardware assignment. The paragraph also specifies alternative input-output areas.

The construct of this paragraph has two options which are:

Option 1:

```
[ FILE-CONTROL.

SELECT [OPTIONAL] file-name-1 ASSIGN TO hardware-name-1

 [      { FILE }  ]    [ { WORK      } ]  [ FOR MULTIPLE REEL ]
 [ BY   { AREA }  ]    [ { PROCESSOR } ]
 [      { NN   }  ]

 [ { NO TRANSLATION              } ]  [ { NO BACKUP } ]     [ FORM ]
 [ { TRANSLATION NON-STANDARD    } ]  [ { BACKUP    } ]

 [ SAVE ]   [ RESERVE    { NO        }  [ ALTERNATE    { AREA  } ] ]
                         { integer-1 }                 { AREAS }

 [ { FILE-LIMIT IS   }  { literal-1   }  { THRU    }  { END         }
 [ { FILE-LIMITS ARE }  { data-name-1 }  { THROUGH }  { literal-2   } ...
                                                      { data-name-2 }

      [ { literal-m   }  { THRU    }  { literal-n   } ] ]
      [ { data-name-m }  { THROUGH }  { data-name-n } ] ]
[                   { RANDOM     } ]
[ ACCESS MODE IS    { SEQUENTIAL } ]  [ACTUAL KEY IS data-name-3]

  [PROCESSING MODE IS SEQUENTIAL]

                                                              ]
[                { KEY IS   }                                 ]
[ SYMBOLIC       { KEYS ARE }  data-name-4 [data-name-5] ... ] ... ] .
                                                              ]
```

Option 2:

```
[
[ FILE-CONTROL.
[
  SELECT sort-file-name ASSIGN TO SORT DISK.    ]
                                                ]
```

Files used in a program must be the subject of only one SELECT statement. If it is to be OPENed INPUT-OUTPUT or I-O, it must be present in the MCP disk directory.

The word OPTIONAL must be used in the SELECT statement whenever an input file can be omitted during certain operational circumstances.

The ASSIGN clause must be used in order for the MCP to associate the file with a hardware peripheral component. The allowable hardware-name entries are:

ATT-8A1
B-500
B-2500
B-3500
B-9350
B-9352
DCT-2000
DISK (or DISC)
DISK SHARED
DISPLAY-UNIT

IBM-1030
IBM-1050
LISTER
O-L-BANKING
PRINTER
PT-PUNCH
PT-READER
PUNCH
READER
SORTER

SPO
TAPE (7 or 9 channel, MCP to assign)
TAPE-PE (Phase encoded)
TAPE-7 (7 channel only)
TAPE-9 (9 channel only)
TC-500
TC-700
TOUCH-TONE
TT-28
TWX

Automatic changing of hardware device at object program run time can be accomplished by the systems operator performing an "IL" control message reflecting the "changed-to" peripherals channel and unit number.

The BY clause is applicable to files ASSIGNed TO DISK only. It is also applicable when the DISK system contains more than one electronic unit (EU's). If the BY clause is omitted, files are physically assigned to disk as space becomes available (from low to high).

The FILE clause specifies the files to be distributed among the electronic units by file number (i.e., order of appearance of the FD).

The AREA option specifies that file-name-1 is to be distributed among the electronic units by area (i.e., as defined in the FILE CONTAINS XX BY XX clause).

The NN option indicates that file-name-1 is to be assigned to the electronic unit specified by integer NN.

The WORK option specifies to the MCP that the SELECTed disk file is to be used as a work file and that the MCP must insert the program's mix-number in the second and third characters of the work file's file-ID, thus creating a unique file-name at object run time. The use of this option allows multiprocessing of the same program without creating duplicate file ID's for commonly used work files.

The PROCESSOR option will specify the ability to have multiple systems sharing common disk. The MCP will insert the processor number in the fifth character of the file ID. The PROCESSOR option is not yet available in the MCP.

The MULTIPLE REEL option is for documentation only. This feature is provided automatically by the MCP.

The NO TRANSLATION clause is used to cause a bypass of the hardware translator of data transfers between hardware-name and internal core memory.

The TRANSLATION NON-STANDARD clause applies to remote devices which are capable of transmitting different codes for upper and lower case characters. If this clause is specified, all data characters transmitted from the device are translated to upper case EBCDIC codes before they are moved to the object program's record area.

The BACKUP option will cause printer output files to be placed on a printer backup tape or disk file for subsequent printing. The BACKUP option will cause punch output files to be placed on punch backup disk files for subsequent punching.

The NO BACKUP option will prevent the file from going to printer backup automatically when the MCP's printer backup option is set "ON" and a Line Printer is not available. This file may be manually assigned to printer backup by the operator with an "OU" or "OUDK" message.

Use of the FORM option with printer files, will cause the program to halt and a MCP message to be printed declaring the need for special forms to be loaded in the Line Printer.

It is recommended that a STOP literal be executed just prior to a STOP RUN if the FORM option is used. This will allow the operator sufficient time to remove the special forms before the printer is released back to the MCP. Without a temporary halt, there is a possibility that another job placed in the mix may start printing on that same printer.

The SAVE option will cause file-name-1 to be CLOSEd WITH LOCK by the MCP if file-name-1 is still OPEN at End-of-Job. If the SAVE option is omitted, then the standard MCP action will be invoked, that is, file-name-1 will be automatically CLOSEd (but not LOCKed) if it is still OPEN at object End-of-Job.

The RESERVE clause allows a variation of the number of input or output physical record buffers to be supplied by the compiler. Each ALTERNATE AREA reserved requires additional memory to be utilized in the compiled object program and will be the size of a physical record as defined in the FD statement of the DATA DIVISION for that specific file. If a SEEK or FILL statement is used in a program, then a RESERVE 1 ALTERNATE AREA clause must be specified. The RESERVE clause is not applicable to SORTER files.

No alternate areas are reserved when the NO option is specified or if the entire option is omitted.

The MCP will keep track of passing record data to or from the buffer and record work area if the dollar sign ($) card specifies MCPB, otherwise the compilers will supply automatic object program code to accomplish this function, thus resulting in a significant increase in object program speed at a cost in users core. The programmer uses READ or WRITE with no concern of buffering action in either case.

The FILE-LIMIT clause is invalid if specified for a sort file description (SD) entry. The FILE-LIMIT clause for input and output files associated with the SORT verb will not be effective when executing the SORT unless there is an INPUT and/or OUTPUT PROCEDURE declared.

The FILE-LIMIT clause specifies that:

a. For SEQUENTIAL access, logical records are obtained from, or placed sequentially in, the disk storage file by the implicit progression from segment to segment. The AT

END imperative statement of a READ statement is executed when the logical end of the last segment of the file is reached and an attempt is made to READ another record. The INVALID KEY clause of a WRITE statement is executed when the end of the last segment is reached and an attempt is made to WRITE another record. The END option specifies that the compiler is to determine the upper limit of an existing file.

b. For RANDOM access, logical records are obtained from, or placed randomly in, the disk storage file within the specified FILE LIMIT. The contents of ACTUAL KEY not within the specified limit will cause the execution of the INVALID KEY branch in the READ and the WRITE statements.

In the FILE-LIMIT clause, each pair of operands associated with the key word THRU represents a logical segment of a file. The logical beginning of a disk storage file is considered to be that address represented by the first operand of the FILE-LIMIT clause; the logical end is considered to be that address as specified by the last operand of the FILE-LIMIT clause.

In a FILE-LIMIT series, SEQUENTIAL records are accessed in the order in which they are specified. For example:

```
FILE-LIMITS 1 THRU 5, 10 THRU 12, 3 THRU 7
```

This example will result in the sequential access of records 1, 2, 3, 4, 5, 10, 11, 12, 3, 4, 5, 6 and 7 in that order.

For the ACCESS MODE SEQUENTIAL clause, the disk storage records are obtained or placed sequentially. That is, the next logical record is made available from the file on a READ statement execution, or a specific logical record is placed into the file on a WRITE statement execution. The ACCESS MODE SEQUENTIAL clause is assumed if ACCESS MODE RANDOM is not specified.

If the ACCESS MODE RANDOM clause is specified, the ACTUAL KEY entry must be used.

Values of the ACTUAL KEY data-name-3 are controlled by the programmer, including any execution of the USE FOR KEY CONVERSION statement. The value may range from 1 to n, where n equals the number of records in the file or as reflected by the FILE-LIMITS clause. The ACTUAL KEY signifies the relative position of a record within the file and is equated to a data-name at any level which is defined with a PICTURE of 9(8) COMPUTATIONAL. ACTUAL KEY is not used for ACCESS MODE SEQUENTIAL files.

The ACTUAL KEY data-name-3 must be declared as PICTURE 9(6) COMPUTATIONAL if referencing data-communications files when the WRITE-TRANS-READ or the WRITE-READ-TRANS verbs are used to communicate between computers.

The ACTUAL KEY clause is not applicable to the SORTER.

The ACTUAL KEY clause is required for the LISTER and data-name-3 must be declared as a PICTURE 9(4) COMPUTATIONAL. Data-name-3 must contain the unit and tape designations to control printing on the LISTER.

The PROCESSING MODE IS SEQUENTIAL clause denotes that disk file records are to be available for processing by the object program in the order in which they are sequentially accessed from segment to segment.

The SYMBOLIC KEY entry is only for documentational purposes. The conversion of SYMBOLIC KEY data-names to the ACTUAL KEY data-name must be specified either by procedural statements preceding the SEEK statements or by means of a USE FOR KEY CONVERSION section in the PROCEDURE DIVISION.

All integers must be of positive values.

File-name-1 must be unique in the first six characters if the use of an MCP label equation card is anticipated for non-disk storage files.

The sort-file-name in Option 2 is the SD level file-name to be used by the SORT verb.

I-O-CONTROL.

The function of this paragraph is to specify memory area, to be shared by different files during object program execution and the point in time that a rerun procedure is to be established.

The construct of this paragraph is:

```
[ I-O-CONTROL.

  [RERUN EVERY integer-1 RECORDS OF file-name-1] ...  [ SAME

[ { RECORD } ] AREA FOR file-name-2 file-name-3 [file-name-4] ... ]
[ { SORT   } ]

 [ MULTIPLE FILE TAPE "multi-file-id" CONTAINS file-name-list

  [POSITION  integer-2 ...]    ...  ]

 [ APPLY      [ { MICR     } ]    [ ALTERNATING    { AREA  } ]
              [ { OCR      } ]                     { AREAS }
              [ { MICR-OCR } ]

 [ WITH [NO-FORMAT] [NO-ERRORS] ]

  [END-TRANSIT]  ON  file-name [...]   .  ] ]
```

The I-O-CONTROL paragraph name may be omitted from the program if the paragraph does not contain any of the clause entries.

The RERUN clause sets up a communication with the MCP to create control procedures whereby an operational program encountering a malfunction can be restarted at the last RERUN control point instead of restarting from the beginning of the program. Integer-1 records cannot exceed 99999.

The SAME AREA clause saves memory space in the object program due to the fact it allows more than one file to share the same file area, associated ALTERNATE, and disk file header areas. As a result, only one of the files sharing the SAME AREA can be OPEN at one time. The length of each area will be determined by the file with the largest record and/or block size. When the SAME SORT AREA is specified it will be considered as being for documentation only due to the unique method of implementing the SORT function. All SORT files make use of all memory area within a program containing the SORT verb.

When the RECORD option of the SAME AREA clause is used, only the record area is shared and the associated alternate areas for each file remain independent. In this case, any number of the files sharing the same record area may be OPEN at one time, but only one of the records can be processed at a time.

The use of the RECORD option may decrease the physical size of a program as well as increase the speed of the object program. To illustrate this point, consider file maintenance. If the SAME RECORD AREA is assigned to both the old and new files, a MOVE will be eliminated which transfers each record from the input to the output area. The records do not have to be defined in detail for both files. Definition of a record within one file and the simple inclusion of an 01 level entry for the other file will suffice. Because these record areas are in fact in the same core location, one set of data names is sufficient for all processing requirements without requiring qualification.

The MULTIPLE FILE clause specifies that two or more files are resident on one magnetic tape. All files resident on a multi-file tape, which are required in a program, must be represented in the source program by a SELECT statement and an FD entry for each file. The file-name-list entries do not have to be in the sequence in which they appear on the tape, nor in the sequence of each FD in which they appear in the source program. However, the MCP will go to the very next file on tape, check the label and if not the file for processing, the MCP will rewind the tape and commence looking for it from the beginning of the tape.

All file-names in a single MULTIPLE FILE clause are implied as utilizing the SAME AREA. The use of SAME AREA would therefore be redundant. The "multi-file-id" is the file-name contained in the physical tape label of a magnetic tape containing multi-files. File-name-list is a series of FD file-names in the program indicated as residing on the multi-file-tape. Multi-files, or any file contained within the file may be OPTIONAL. The POSITION clause is for documentation only.

The APPLY clause is specified for SORTER only.

If no MICR, OCR, or MICR-OCR option is specified, MICR is assumed.

No alternating areas are assumed if ALTERNATING is not specified. ALTERNATING applies to OCR only.

Formatting is assumed if NO-FORMAT is not specified (see READ, section 9).

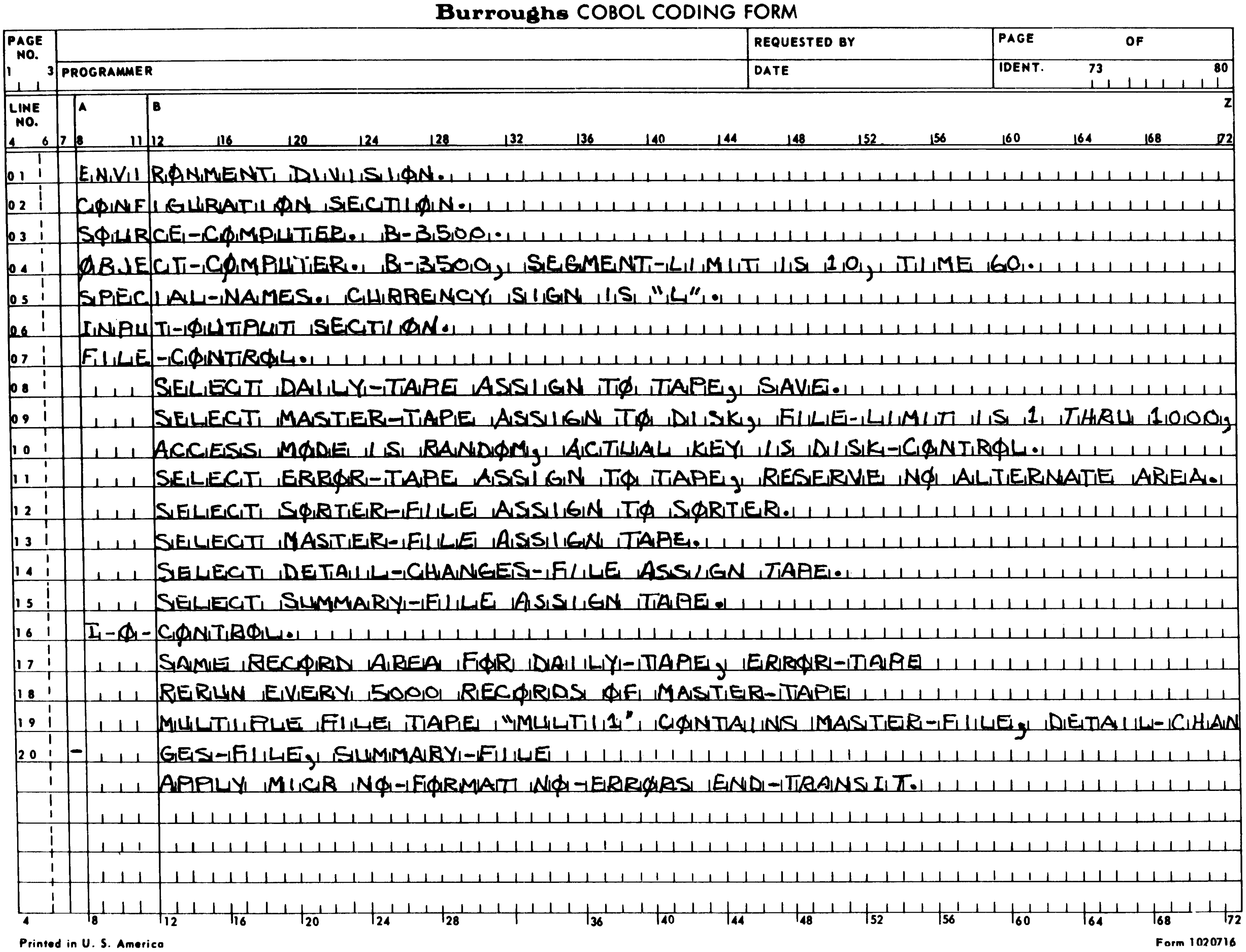
An error branch is assumed if NO-ERRORS is not specified.

The End-of-Document is assumed if END-TRANSIT is not specified.

The I-O-CONTROL paragraph can have only a terminating period.

CODING THE ENVIRONMENT DIVISION.

An example of ENVIRONMENT coding is provided in figure 3-1.



Burroughs COBOL CODING FORM

PAGE NO. 1 3 | PROGRAMMER | REQUESTED BY | DATE | PAGE OF | IDENT. 73 80

LINE NO. 4 6 | 7 | A 8 11 | B 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72 Z

```
01    ENVIRØNMENT DIVISIØN.
02    CØNFIGURATIØN SECTIØN.
03    SØURCE-CØMPUTER. B-3500.
04    ØBJECT-CØMPUTER. B-3500, SEGMENT-LIMIT IS 10, TIME 60.
05    SPECIAL-NAMES. CURRENCY SIGN IS "L".
06    INPUT-ØUTPUT SECTIØN.
07    FILE-CØNTRØL.
08        SELECT DAILY-TAPE ASSIGN TØ TAPE, SAVE.
09        SELECT MASTER-TAPE ASSIGN TØ DISK, FILE-LIMIT IS 1 THRU 1000,
10        ACCESS MØDE IS RANDØM, ACTUAL KEY IS DISK-CØNTRØL.
11        SELECT ERRØR-TAPE ASSIGN TØ TAPE, RESERVE NØ ALTERNATE AREA.
12        SELECT SØRTER-FILE ASSIGN TØ SØRTER.
13        SELECT MASTER-FILE ASSIGN TAPE.
14        SELECT DETAIL-CHANGES-FILE ASSIGN TAPE.
15        SELECT SUMMARY-FILE ASSIGN TAPE.
16     I-Ø-CØNTRØL.
17        SAME RECØRD AREA FØR DAILY-TAPE, ERRØR-TAPE
18        RERUN EVERY 5000 RECØRDS ØF MASTER-TAPE
19        MULTIPLE FILE TAPE "MULTI1" CØNTAINS MASTER-FILE, DETAIL-CHAN
20 -      GES-FILE, SUMMARY-FILE
          APPLY MICR NØ-FØRMAT NØ-ERRØRS END-TRANSIT.
```

4 8 12 16 20 24 28 36 40 44 48 52 56 60 64 68 72

Printed in U. S. America

Form 1020716

Figure 3-1. Coding the ENVIRONMENT DIVISION

[redacted]

[redacted]

SECTION 4
DATA DIVISION

GENERAL.

The third part of a COBOL source program is the DATA DIVISION which describes all data that the object program is to accept as input, and to manipulate, create, or produce as output. The data to be processed falls into three categories:

a. Data which is contained in files and enters or leaves the internal memory of the computer from a specified area or areas.

b. Data which is developed internally and placed into intermediate storage, or into a specific format for output reporting purposes.

c. Constants which are defined by the programmer.

DATA DIVISION ORGANIZATION.

The DATA DIVISION is subdivided into two sections:

a. The FILE SECTION which defines the contents of data files which are to be created or used by an external medium. Each file is defined by a file description, followed by a record description or a series of file-related record descriptions.

b. The WORKING-STORAGE SECTION describes records, constants, and non-contiguous data items which are not part of an external data field, but are developed and processed internally.

DATA DIVISION STRUCTURE.

The general structure of the DATA DIVISION is as follows:

```
DATA DIVISION.
[FILE SECTION.]
[FD file-name-1 . . .  .]
```

```
    [01 record-name-1 .]
    [02 data-name-1 . . .] .
    [02 . . .] .
    [03 data-name-2 . . .] .
    [01 record-name-2 .]
  [SD file-name-2 .]
  [WORKING-STORAGE SECTION.]
    [77 data-name-3 . . . ] .
    [77 data-name-4 . . . ] .
    [01 record-name-3 .]
    [02 data-name-5 . . .] .
    [02 data-name-6 . . .] .
    etc.
    [01 record-name-4 .]
    etc.
```

NOTE

The DATA DIVISION cannot exceed 100,000 COMPUTATIONAL digits or 50,000 DISPLAY Characters.

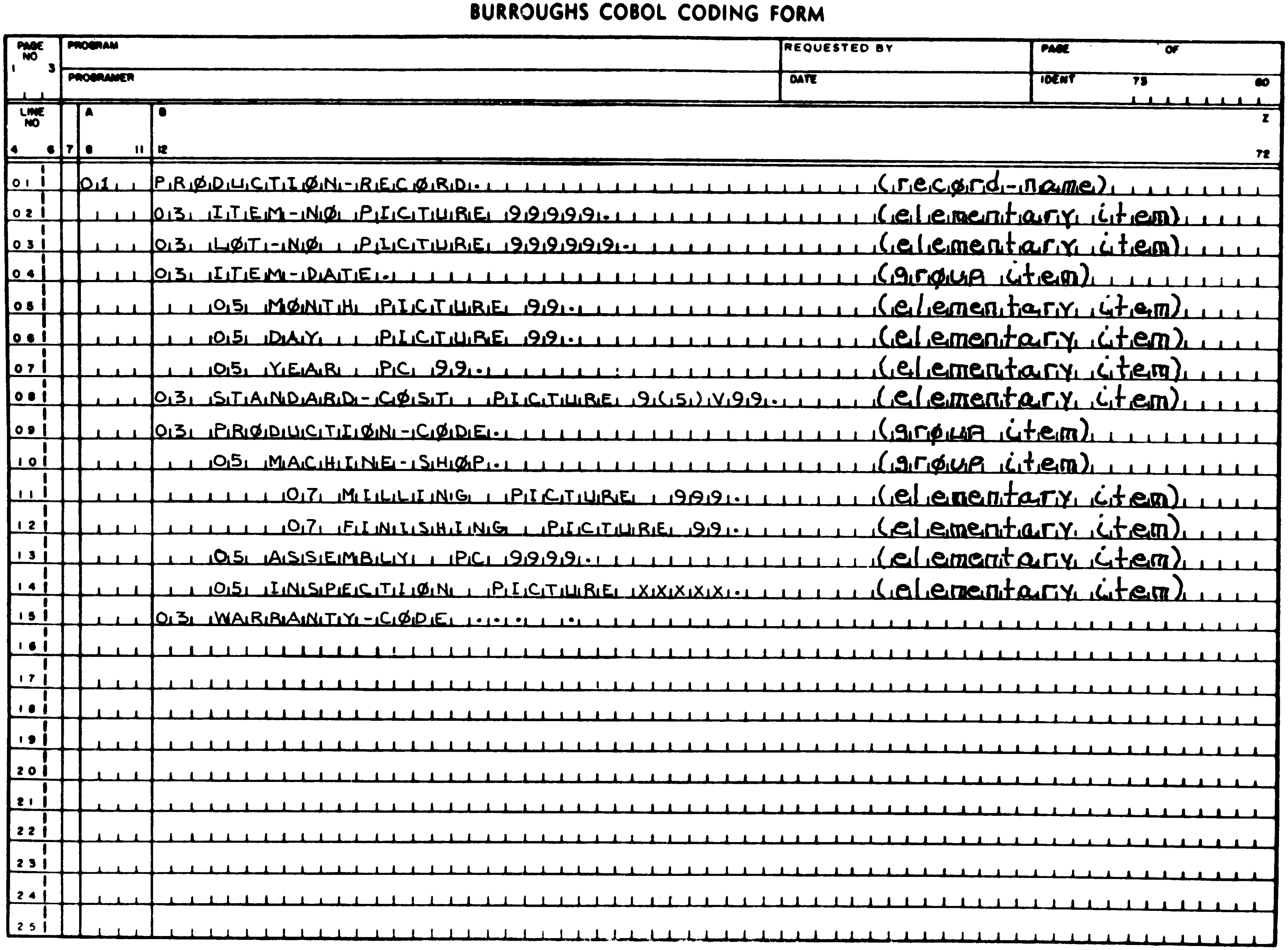
RECORD DESCRIPTION STRUCTURE.

A Record Description consists of a set of data description entries which describe the elements within a particular record. Each data element consists of a level-number followed by a data-name, followed by a series of independent clauses, as required. A Record Description has a hierarchical structure and therefore the clauses used with an entry may vary considerably, depending upon whether or not it is followed by subordinate elementary entries.

LEVEL-NUMBER CONCEPT.

The level-number shows the hierarchy of data within a logical record. In addition, it is used to identify entries for Condition-Names, non-contiguous constants, Working-Storage items, and the RENAMES clause.

Each record of a file begins with the level-number 01 (which may also be shown as 1). This number is reserved for the record-name only, as the most-inclusive grouping for a record. Less-inclusive groupings are given higher numbers, but not necessarily successively. The numbers can range up to 49. Figure 4-1 illustrates the use of level within a record.



BURROUGHS COBOL CODING FORM

PAGE NO 1 3 | PROGRAM | REQUESTED BY | PAGE OF

PROGRAMER | DATE | IDENT 73 80

LINE NO 4 6 | 7 | A 8 11 | B 12 Z 72

```
01  01  PRØDUCTIØN-RECØRD.                            (recørd-name)
02      03 ITEM-NØ PICTURE 99999.                     (elementary item)
03      03 LØT-NØ  PICTURE 999999.                    (elementary item)
04      03 ITEM-DATE.                                 (grøup item)
05         05 MØNTH PICTURE 99.                       (elementary item)
06         05 DAY   PICTURE 99.                       (elementary item)
07         05 YEAR  PC 99.                            (elementary item)
08      03 STANDARD-CØST  PICTURE 9(5)V99.            (elementary item)
09      03 PRØDUCTIØN-CØDE.                           (grøup item)
10         05 MACHINE-SHØP.                           (grøup item)
11               07 MILLING  PICTURE  999.            (elementary item)
12               07 FINISHING  PICTURE 99.            (elementary item)
13         05 ASSEMBLY  PC 9999.                      (elementary item)
14         05 INSPECTIØN  PICTURE XXXXX.              (elementary item)
15      03 WARRANTY-CØDE ....  .
16
17
18
19
20
21
22
23
24
25
```

Figure 4-1. Coding of Level-Number

Revised 9/28/70
by PCN 1033099-004

For an item to be elementary, it can not have subordinate levels. Therefore, the smallest element of a data description is called an elementary item. In figure 4-1, MONTH, DAY, YEAR, MILLING, and FINISHING are elementary items. Since ITEM-NO, LOT-NO, STANDARD-COST, ASSEMBLY, INSPECTION, and WARRANTY-CODE do not have subsidiary clauses, they also represent elementary items.

A level that has further subdivisions is called a group item. In figure 4-1, ITEM-DATE, PRODUCTION-CODE, and MACHINE-SHOP represent items on a group level. A group is defined as being composed of all group and elementary items described under it. A group item ends when a level-number of equal or lower numeric value than the group item itself is encountered. In figure 4-1, group item PRODUCTION-CODE ends with INSPECTION. A group item can only consist of a level-number and a data-name followed by a period. COBOL defines all group items to be alphanumeric and will be byte aligned by the compiler. The FILLER ADDED message will appear where such alignment has taken place. Apart from level-numbers 01 through 49, three additional level-numbers exist in COBOL. These are numbers 66, 77, and 88. They represent level-numbers within RENAMES, WORKING-STORAGE, and Condition-Name entries respectively.

To reiterate, a level-number is the first required element of each record and data description entry. In value it can range from 01 through 49 (1, 2, etc. is also permissible), plus special numbers of 66, 77, and 88. It is important to remember that multiple level 01 entries of a given File Description of the File Section represent implicit redefinition of the same core area.

QUALIFICATION.

The data-names of the DATA DIVISION need not be unique as long as the parent item of that data-name is unique in itself. Qualification is accomplished by following the data-name to be qualified with either IN or OF and the qualifying data-name, record-name or file-name. In the example below, all item descriptions (except the data-name PREFIX) are unique. In order to refer to either PREFIX item, qualification must be used. Otherwise, if reference is made to PREFIX only, the compiler would not know which of the two is desired. Therefore, in order to move the contents of PREFIX into PREFIX of the other, the PROCEDURE DIVISION must be coded with one of the following sentences:

```
a.  MOVE PREFIX OF ITEM-NO TO PREFIX IN CODE-NO.
b.  MOVE PREFIX OF ITEM-NO TO PREFIX IN MASTER-FILE.
c.  MOVE PREFIX OF TRANSACTION-TAPE TO PREFIX IN CODE-NO.
d.  MOVE PREFIX OF TRANSACTION-TAPE TO PREFIX IN MASTER-FILE.
```

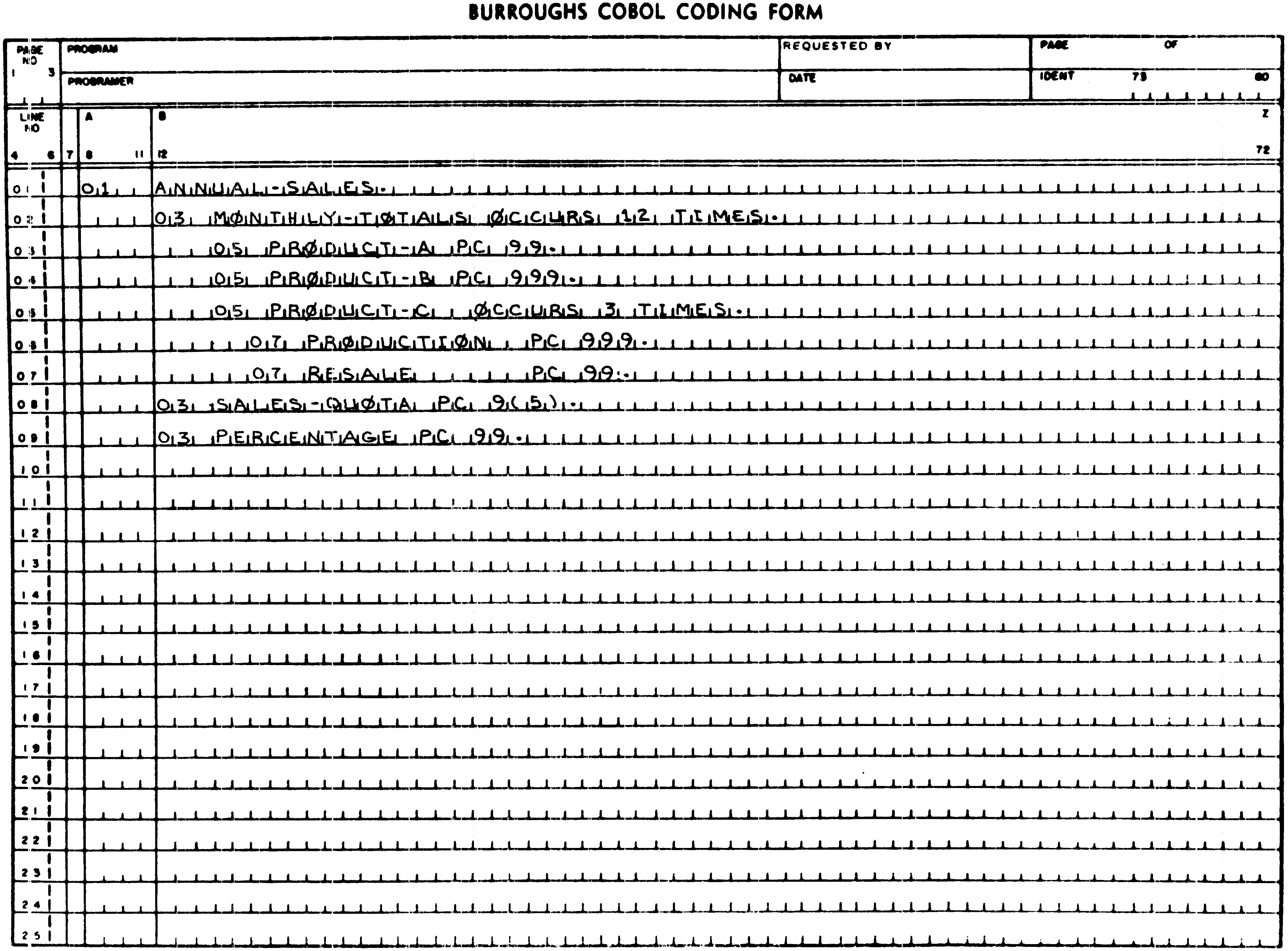
EXAMPLE:

```
01 TRANSACTION-TAPE .....      01 MASTER-FILE .....
   03 ITEM-NO .....               03 CODE-NO .....
      05 PREFIX .....                05 PREFIX .....
      05 CODE .....                  05 SUFFIX .....
   03 QUANTITY .....              03 DESCRIPTION .....
```

TABLES.

Frequently, the need arises to describe data which appears in a table or an array. For example, an annual sales total record might have to be broken down by months. In order to accomplish this, January sales would have to be referred to by a given data-name, February sales by another, etc. By using the OCCURS clause, the same result can be obtained without the need for 12 different data-names. Figure 4-2 illustrates how the OCCURS clause may be used in order to have the compiler build a table of twelve elements, each having a structure like MONTHLY-TOTALS. The first element will be known as 1 of the table, the second as 2, etc. The technique of referring to elements within a table or an array is known as subscripting.

The OCCURS clause may appear at any level except the 01 level which is reserved for record-names. For more detailed information, refer to the OCCURS clause.



BURROUGHS COBOL CODING FORM

PAGE NO 1 3 | PROGRAM | REQUESTED BY | PAGE OF

PROGRAMER | DATE | IDENT 73 80

LINE NO 4 6 | 7 | A 8 11 | B 12 | Z 72

```
01  01  ANNUAL-SALES.
02      03 MØNTHLY-TØTALS ØCCURS 12 TIMES.
03          05 PRØDUCT-A PC 99.
04          05 PRØDUCT-B PC 999.
05          05 PRØDUCT-C  ØCCURS 3 TIMES.
06              07 PRØDUCTIØN  PC 999.
07              07 RESALE      PC 99.
08      03 SALES-QUØTA PC 9(5).
09      03 PERCENTAGE PC 99.
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
```

Figure 4-2. Coding of Multi-Dimensioned Table

The repetition of data elements applies to all subordinate fields. OCCURS may be nested to describe tables of more than one dimension by applying an OCCURS clause to a subordinate name. The COBOL compiler permits tables of up to three dimensions.

SUBSCRIPTING.

When a data-name OCCURS more than once, the particular element desired within the array is referred to by using subscripts. The subscripts follow the data-name representing the array in a COBOL statement. A space may separate the data-name and the subscript bounded by parentheses. A subscript may be either a numeric literal or a data-name. A data-name being used as a subscript may not be subscripted. If the value of a subscript is changed in a series (e.g., MOVE A (B) to C (B), B, D (B).) the subscript (for D (B) in this case) is not re-evaluated.

In order to reference the first occurrence of MONTHLY-TOTALS of figure 4-2, one may write: ...MONTHLY-TOTALS (data-name), where data-name must contain a 1, or MONTHLY-TOTALS (1).

If data-name INCREMENTER is used to refer to the desired element in a table in terms of the sample illustration, MONTHLY-TOTALS (INCREMENTER) would have to be written. In this case, the INCREMENTER would have to contain that value which represents the desired element. If a specific RESALE item within a given month is again desired, RESALE (INCREMENTER, CODE-X) would have to be written. CODE-X is a data-name that can have a value of 1, 2, or 3, depending on which of the levels is required.

At that point in time when a data-name is used for subscripting purposes, its value must be greater than zero but not greater than the value shown in the corresponding OCCURS clause*.

Where qualification and subscripting are to be used simultaneously, the qualification has to be shown first, followed by the subscripting.

* The generated object code will not check the validity of data-name values used for subscripting or indexing and undefined results will occur should the program reference a subscripted data-name or an index-data-name containing a value of zero, or a value above the defined subscript or index range as reflected in the OCCURS clause pertaining to that item.

FILE DESCRIPTION.

FILE SECTION.

This section contains descriptions of the files used by the object program.

FILE DESCRIPTION

The function of this paragraph is to furnish information to the compiler concerning the physical structure, identification, and record names pertaining to a given file.

The construct of this paragraph contains four options:

Option 1:

```
FD  file-name  COPY  "library-name".
```

Option 2:

```
FD file-name-1  [ RECORDING MODE IS { STANDARD     } ]
                [                   { NON-STANDARD } ]

FILE CONTAINS integer-1 [BY integer-2] RECORDS

[ BLOCK CONTAINS [integer-3 TO]  integer-4   { RECORDS    } ]
[                                            { CHARACTERS } ]

[ RECORD CONTAINS [integer-5 TO] integer-6  CHARACTERS ]

[ LABEL  { RECORD IS    }  { OMITTED      } ]
[        { RECORDS ARE  }  { STANDARD     } ]
[                          { USASI        } ]
[                          { NON-STANDARD } ]

[{ VA    }  OF  { ID             }  IS  { "literal-1"  }
[{ VALUE }      { IDENTIFICATION }      { data-name-1  }

      [SAVE-FACTOR IS literal-2]  ]

[ DATA  { RECORD IS   }  data-name-2  [data-name-3 ...]  ]
[       { RECORDS ARE }                                  ]
```

Option 3:

```
SD  sort-file-name  COPY  "library-name".
```

Option 4:

```
SD sort-file-name

 FILE CONTAINS integer-1 [BY integer-2] RECORDS

[ RECORD CONTAINS [integer-3 TO integer-4 CHARACTERS] ]

[ BLOCK CONTAINS [integer-5 TO] integer-6 { RECORDS    } ]
                                          { CHARACTERS }

[ DATA { RECORD IS   } data-name-1 [data-name-2] ... ] .
       { RECORDS ARE }
```

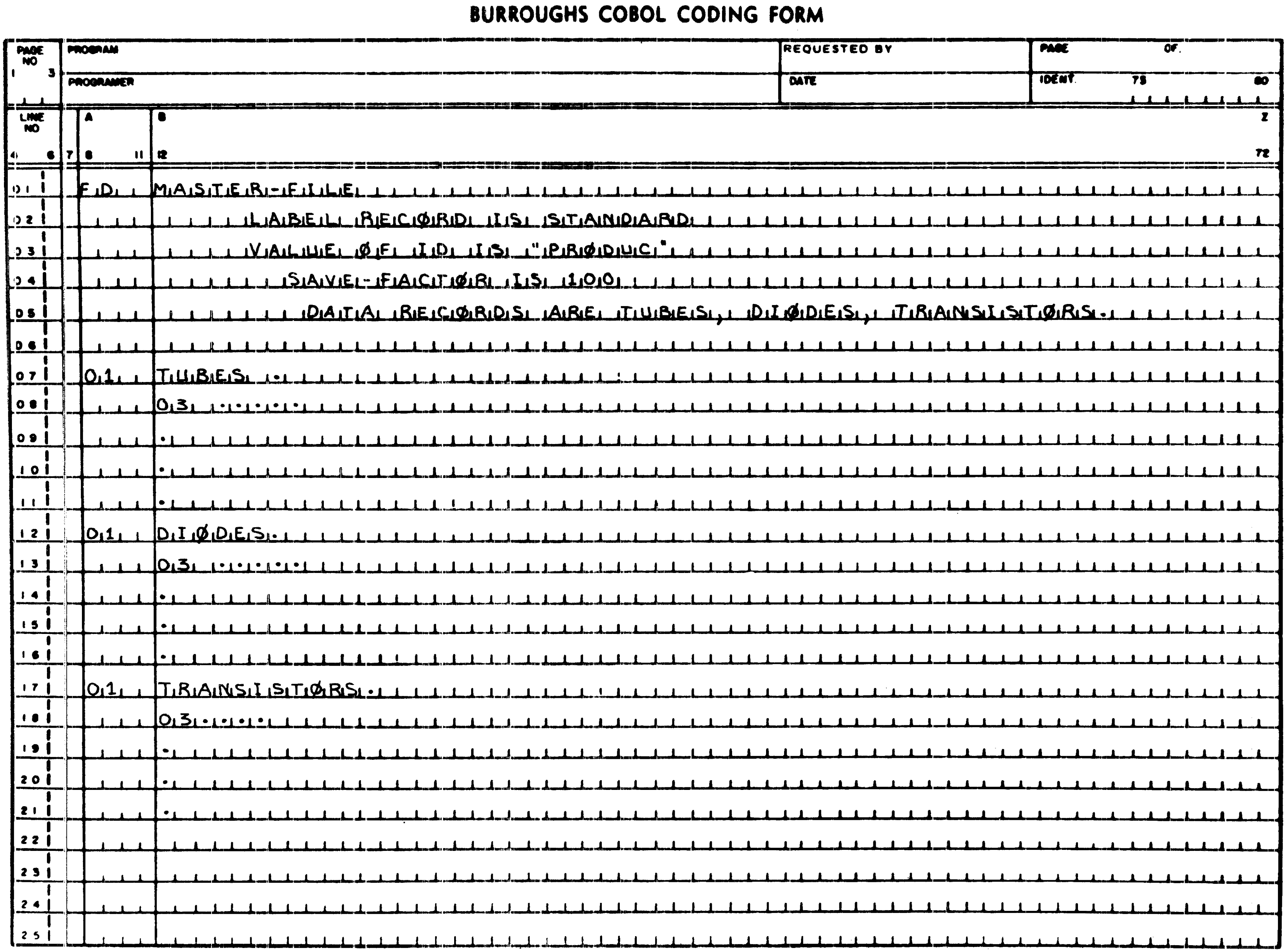
The level indicator, FD and SD identify the beginning of a File Description or a Sort File Description and must precede the file statement. Both entries must commence under Area A of the coding form. Only one period is allowed in the entry and it must follow the last used clause.

Options 1 and 3 can be used when the Systems library contains the library-name entry, otherwise, Option 2 and/or 4 must be used.

In many cases, the clauses within the File Description, or Sort File Description sentence are optional. Each clause is discussed in detail.

NOTE

Figure 4-3 illustrates the use of the File Description sentence followed by data record entries. It is further noted that the three 01 levels implicitly redefine the record area and that the DATA RECORDS clause is treated by the compiler as being documentation only and does not cause an explicit redefinition of the area.



BURROUGHS COBOL CODING FORM

PAGE NO 1 3 | PROGRAM | REQUESTED BY | PAGE OF

PROGRAMER | DATE | IDENT. 73 80

LINE NO 4 6 | 7 | A 8 11 | B 12 | Z 72

```
01   FD  MASTER-FILE
02                LABEL RECØRD IS STANDARD
03                VALUE ØF ID IS "PRØDUC"
04                  SAVE-FACTØR IS 100
05                    DATA RECØRDS ARE TUBES, DIØDES, TRANSISTØRS.
06
07    01  TUBES .
08        03 .......
09        .
10        .
11        .
12    01  DIØDES.
13        03 .......
14        .
15        .
16        .
17    01  TRANSISTØRS.
18        03.....
19        .
20        .
21        .
22
23
24
25
```

Figure 4-3. Coding of FD and DATA RECORDS

BLOCK.

The function of this clause is to specify the size of a physical record (block).

The construct of this clause is:

```
[ BLOCK CONTAINS [integer-1 TO]   integer-2  { RECORDS    } ]
                                             { CHARACTERS }
```

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

The clause is required if the block contains more than one logical record.

When only integer-2 is used, it will represent logically blocked, fixed-length, records if its value is other than 1. When the integer-1 TO integer-2 option is used, it will represent the minimum to maximum size of the physical record and indicates the presence of blocked variable-length records. Integer-1 is for documentation purposes only.

The maximum value of the integer used in this clause is shown in table 4-1 and refers to the number of characters in a block.

The word CHARACTERS is an optional word in the BLOCK clause. Whenever the key word RECORDS is not present, the integers represent characters and must be modulo 2.

For object program efficiency, the use of blocked records is recommended. The physical size of the block should be as large as possible depending on memory availability.

Blocks of records are READ into the input record buffer area by the MCP. When the dollar sign ($) card reflects "MCPB", the MCP will deliver each record to the programs record work-area as required by every explicit READ command. Omission of the MCPB option will cause the compiler to create object code for programmatic unblocking of records at a considerable increase in speed. In either case, unblocking of records is of no concern to the programmer.

Table 4-1
Maximum Value of Integers

I/O Medium	Maximum Block Size - Characters
READER	80
PUNCH	80
TAPE	Limited only by the amount of memory available.
DISK	Limited only by the amount of memory available.
PRINTER	One print line.
PT-READER	Limited only by the amount of memory available.
PT-PUNCH	Limited only by the amount of memory available.
SORTER	200
LISTER	44

Every explicit WRITE verb causes compiler generated object code to deliver a record to a files output record buffer area and to accumulate the number of logical records required to create a specified block size before notifying the MCP to write the block. When a file is CLOSEd, the records left in the output buffer area will be written by the MCP before the file is physically CLOSEd. The coding of record area to buffer is automatic and is of no concern to the programmer. The blocking of records by the object program can be inhibited by placing "MCPB" in the dollar sign ($) card, thus assigning the task to the MCP (see section 8) however, speed will be sacrificed for the resultant core savings.

The user must specify the actual size of variable-length records in the first four bytes of each record. This four-character indicator is counted in the physical size of each record.

The BLOCK clause is not applicable to the SORTER, LISTER, PT-PUNCH or PT-READER peripherals.

This clause may be omitted for unblocked files.

DATA RECORDS

DATA RECORDS.

The function of this clause is to document the names of the logical record(s) actually contained within the file being described.

The construct of this clause is:

```
[ DATA   { RECORD IS   }   data-name-2  [data-name-3...] ]
         { RECORDS ARE }
```

This statement is only for documentation purposes. The compiler will obtain this information from 01 level record description entries.

FILE CONTAINS.

The function of this clause is to indicate the number of logical records in a file. This statement is required for disk files, and optional for all other files.

The construct of this clause is:

```
FILE    CONTAINS    integer-1    [BY integer-2]  RECORDS
```

The indicated integers must be positive values.

Integer-1 may not exceed 20 when integer-2 is present. The product of integer-1 BY integer-2 cannot exceed 99,999,999. When integer-1 is used by itself, it cannot exceed 99,999,999.

An entry of FILE CONTAINS 20 BY 9999 RECORDS will notify the MCP to allot 20 separate areas (pages) of disk as each area is programmatically required. The size of each page would be 9,999 logical records in length.

The above technique allows the MCP to efficiently assign file pages as needed, rather than immediately assigning one huge file area during the first operation of the program.

Programmatic usage of the file can either enhance the paging technique or defeat its purpose completely. For example, assume that a RANDOM file at some future date will require a maximum size of 20 x 9999 (199,980) logical records, and that no key conversion formula is used due to the key being a six digit number running from 1 thru 199,980 which exactly fills the key requirement, as is the case in auto license numbers in some states. It could happen that the first twenty records could open up an entire disk module if they were in increments of 9999, which would negate the paging technique completely and thus causes the MCP Disk Directory to recognize the file as being of maximum size, even though only twenty records were processed.

The programmer should utilize the USE FOR KEY CONVERSION technique to programmatically link records to the first page of the file and to try and use up as many record spaces as possible within each page before forcing the MCP to open another.

The following B 2500/B 3500 statistics define the maximum disk file storage area for a given file as being 20 Electronic Units, where each E.U. contains one page of the file on five contiguous disk modules within the control of the E.U. A file page cannot continue from one E.U. onto another, however, file pages continue from module to module within the control of each E.U.

Model 1A Disk Storage		Model 1C Disk Storage
5	Modules of disk per E.U.	5
20	E.U.s per system.	20
100,000	Segment addresses per module.	200,000
500,000	Segment addresses per E.U.	1,000,000
10,000,000	Bytes per module.	20,000,000
50,000,000	Bytes per E.U.	100,000,000

A disk file residing on Model 1C Disk Storage systems cannot contain more logical records than the equivalent of 5,000,000 twenty byte records, blocked 5 per physical record, in a single disk file page and cannot contain more than twenty disk file pages per B 2500/B 3500 system.

EXAMPLE:

5,000,000 records x 20 bytes per record =
100,000,000 bytes (maximum for an E.U.)

100,000,000 bytes per E.U. ÷ 100 byte segments =
1,000,000 segment address per E.U.

The FILE-LIMITS clause, if present, overrides this clause for INVALID KEY and End-of-File checking. INVALID KEY (or AT END) must be present during READ/WRITE operations on files specified with FILE-LIMIT.

LABEL.

The function of this clause is to specify the presence or absence of file label information as the first and last record of an input or output file.

The construct for this clause is:

$$\left[\ \underline{\text{LABEL}}\ \left\{\begin{array}{l}\underline{\text{RECORD}}\ \text{IS}\\ \underline{\text{RECORDS}}\ \text{ARE}\end{array}\right\}\left\{\begin{array}{l}\underline{\text{OMITTED}}\\ \underline{\text{STANDARD}}\\ \underline{\text{USASI}}\\ \underline{\text{NON-STANDARD}}\end{array}\right\}\ \right]$$

The LABEL clause is not applicable to SORTER or LISTER. If this statement is omitted, files will be assumed to contain or to have been created with STANDARD labels.

STANDARD specifies that labels exist for the file or device to which the file is assigned. It also specifies that input and output labels conform to the MCP standards as implemented.

STANDARD, when specified for disk files, indicates that the six character contents of the VALUE or ID clause will be inserted into the disk file header. Should VALUE of ID be omitted, the first six characters FD or SD file-name will be inserted into the disk file header.

OMITTED specifies that the labels do not exist for the specific input file or device to which the file is ASSIGNed. During object program operation, the operator will be queried by the MCP as to where the input is located. The operator must reply with mix-index UL channel/unit control message.

NON-STANDARD indicates, that the files physical magnetic tape label is formatted as an EDP installations own standard label which has been appropriately defined in the System Specification Deck at "cold start" time (see B 2500/B 3500 MCP Reference Manual for specifications relating to Installation Labels).

OMITTED specifies that labels are not to be created for the specific output file ASSIGNed.

The Burroughs Standard B 2500/B 3500 label record serves as both the beginning label record and the ending label record and is comprised of the following parts:

Positions	Field Description
1	Invalid character for card files and blank for other files.
2-8	"LABELbb"
9	Zero.
10-15	"Multiple-file-id" or zeros.
16	Blank.
17	Zero.
18-23	"File-identifier".
24	Blank.
25-27	Reel number within a magnetic tape file.
28-32	Date written (creation date YYDDD).
33-34	Cycle (distinguishing multi-runs of the program).
35-39	Purge-date (YYDDD) at which time the MCP assumes a magnetic tape as "scratch".
40	Sentinel (0 = End-of-File and 1 = End-of-Reel).
41-45	Block count (ending label only).

Positions	Field Description
46-52	Record count (ending label only).
53	Memory dump key (1 = memory dump follows beginning label).
54-58	External magnetic tape library reel number.
59-80	Reserved.
81-	User's portion.

The COBOL compiler will obtain the value of "multiple-file-ID" from the I-O-CONTROL MULTIPLE FILE TAPE clause.

The COBOL compiler will obtain the value of the "file-identifier" from the FD VALUE OF ID IS clause, or if it has been omitted it will be taken from the first six characters of the FD-name.

The initial value of the reel number is preset at 001 and increased as required during operation of the object program.

The value of date written is as maintained on the system for the processing day and converted by the MCP to YYDDD.

The value of cycle is preset to 01. It is desired to run the same program more than once during a given days period, the operator should be given the parameters of a VALUE control statement.

The value of sentinel will be set to 1 at the end of every reel within a file and to zero at the end of the file.

The block count value will contain the number of record blocks on the tape and is written on the output reel's ending label. The MCP will keep count of all blocks read as input and will verify that it has read all blocks by comparing the created total and the block count entry of each reel's ending label.

The record count value will contain the number of logical records on a tape and is written on the output reel's ending label. The

MCP will verify the number of records read in the same manner as for block count.

The memory dump key notifies the MCP to format the output into memory dump notation for the printer.

The external magnetic tape library reel number is initially placed in the label, by a user program, and is permanently maintained by the MCP regardless of the tape status of being a scratch or current data tape. This area of the tape label may be altered by a DECLARATIVE in a user program.

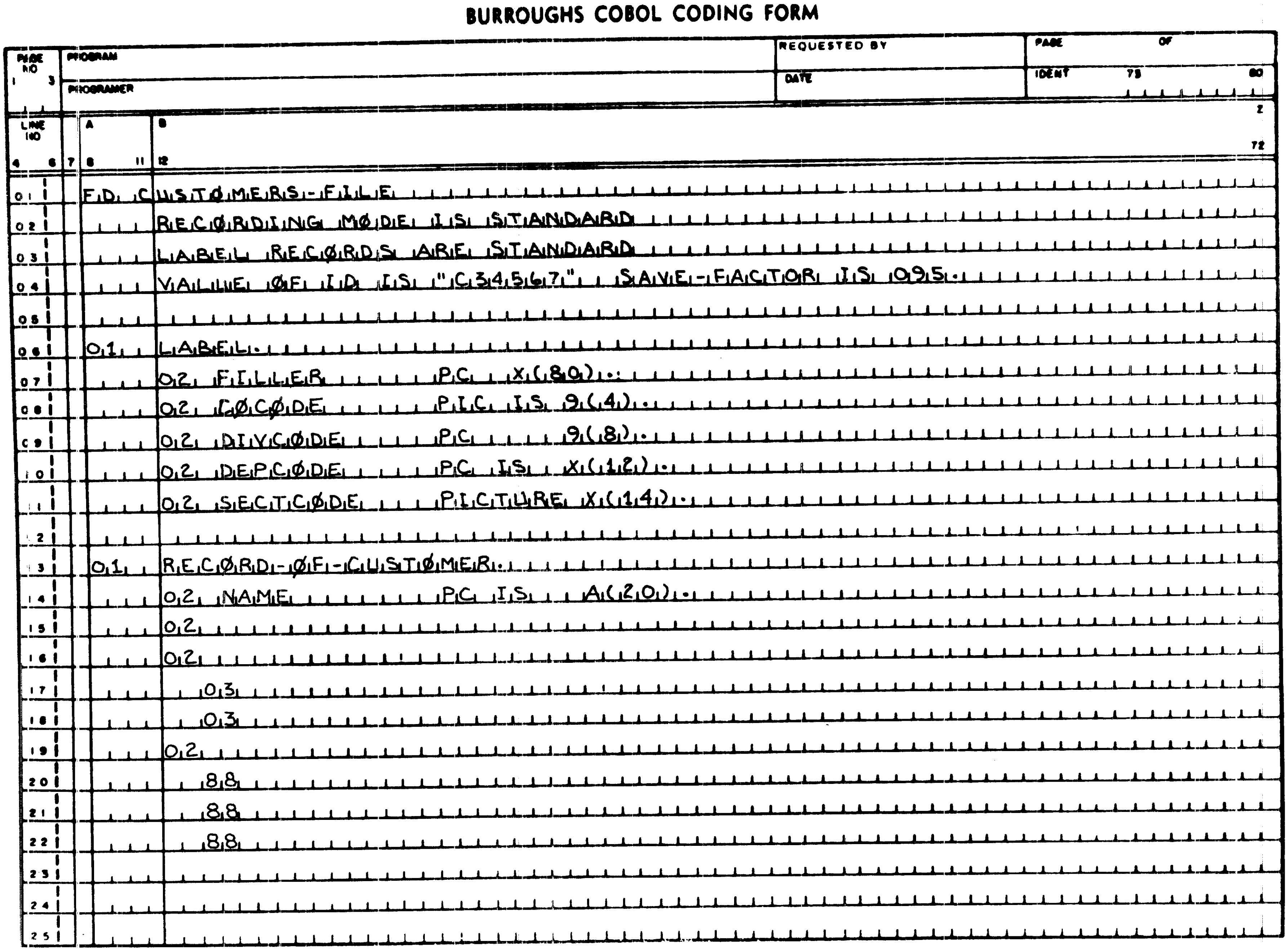
The STANDARD label may contain data designed by the programmer to accommodate miscellaneous information pertinent to magnetic tape files only. This user's portion along with the STANDARD information, or FILLER, must be described within the first 01 record description entry of the file. The compiler recognizes only an entry of 01 LABEL as containing a label description for a given file which precedes an actual record description (see figure 4-4 for an example of Level Coding).

The user's portion of the STANDARD label may be of any length.

The ANSI Standard B 2500/B 3500 label record serves both the beginning label record and the ending label record. Its format is as follows:

Positions	Field Description	
1-3	HDR.	
4	1	
5-13	blank.	
14-21	"file identifier".	The MCP recognizes the middle six characters as the file-ID. Position 1 must be a zero and position 8 must be a blank. E.G., 0AAAAAAΔ.
22-27	"multiple-file-ID".	

Positions	Field Description
28	0 (zero).
29-31	Reel number within a magnetic tape file.
32-35	0001 (file sequence number).
36-39	Blanks (generation number optional).
40-41	Cycle number (generation version number optional).
42-47	bYYDDD (creation date).
48-53	bYYDDD (purge date).
54	Blank (accessability).
55-60	000000 (block count (end label block count)).
61-67	0000000 (record count (end label record count)).
68-72	Physical tape number.
73	B (optional).
74-80	blanks.
81-	User's portion.



BURROUGHS COBOL CODING FORM

PAGE NO 1 3 | PROGRAM | REQUESTED BY | PAGE OF
PROGRAMER | DATE | IDENT 73 80

LINE NO 4 6 | 7 | A 8 11 | B 12 | Z 72

```
       FD  CUSTØMERS-FILE
           RECØRDING MØDE IS STANDARD
           LABEL RECØRDS ARE STANDARD
           VALUE ØF ID IS "C34567"  SAVE-FACTOR IS 095.

       01  LABEL.
           02 FILLER        PC  X(80).
           02 CØCØDE        PIC IS 9(4).
           02 DIVCØDE       PC    9(8).
           02 DEPCØDE       PC IS  X(12).
           02 SECTCØDE      PICTURE X(14).

       01  RECØRD-ØF-CUSTØMER.
           02 NAME             PC IS  A(20).
           02
           02
             03
             03
           02
             88
             88
             88
```

Figure 4-4. Label Coding

RECORD.

The function of this clause is to specify the minimum and maximum variable record lengths. This clause is not applicable to disk files.

The construct of this clause is:

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

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

If integer-1 and integer-2 are indicated, the variable length record technique is utilized.

If only integer-2 is indicated, the compiler will treat the clause as documentational only.

If integer-1 and integer-2 are indicated, they refer to the minimum and maximum size of the variable records to be processed. At least one record description must reflect the maximum size record length as specified in the RECORD CONTAINS clause.

The user must specify the actual size of variable-length records in the first four bytes of each record and the record size must contain an even number of characters (MOD 2). The four-character variable-size indicator is counted in the physical size of each record.

RECORDING MODE.

The function of this clause is to specify the recording mode for peripheral devices where a choice can be made.

The construct for this clause is:

```
[ RECORDING MODE IS  { STANDARD     } ]
                     { NON-STANDARD } 
```

The RECORDING MODE clause is not applicable to SORTER. The SORTER control will translate the 4-bit MICR code from the SORTER into EBCDIC.

STANDARD RECORDING MODE is assumed if this clause is absent from the FD sentence. The MCP automatically checks the parity of input magnetic tapes and will read the tape in the intelligent mode. For this reason, this clause is required only for tapes when the output is to be NON-STANDARD. RECORDING MODE for the card reader is determined at execution time by a Label Card containing ?DATA (EBCDIC as STANDARD) or ?DATAB (BCL as NON-STANDARD).

The MCP will automatically assign STANDARD RECORDING MODE on 9-channel magnetic tape drives if a SELECT clause indicates TAPE, even though the programmer has designated the unit as being NON-STANDARD.

The recording modes for the peripheral devices are provided in table 4-2.

Table 4-2

Recording Modes for Peripheral Devices

Device	Standard	Non-Standard
TAPE-7	Odd Parity	Even Parity
TAPE-9	Odd Parity	-
DISK	Memory Image	-
READER	Documentational Only	Documentational Only
PUNCH	EBCDIC	BCL
PT-READER	BCL	Binary
PT-PUNCH	BCL	Binary
PRINTER	BCL	-
SORTER	-	-
LISTER	BCL	-

VALUE-OF-ID.

The function of this clause is to define the identification value assigned, or to be assigned, to a file of records and to declare the length of time that a file is to be saved.

The construct of this clause is:

```
[ { VA    }  OF  { ID             }  IS  { "literal-1"  } ]
[ { VALUE }      { IDENTIFICATION }      { data-name-1  } ]

     [SAVE-FACTOR IS literal-2]
```

This clause may be used when label records are present in the file being described. If this clause is not present, the VALUE OF ID will be taken from the first six characters of the FD name which must be uniquely constructed so that the MCP will be able to recognize the files. For example:

```
FD OUTPUT-TO-DISK1  }    Would create a VALUE of ID
FD OUTPUT-TO-DISK2  }    as OUTPUT for both files
                         and will cause DUP FILE
                         action by the MCP.
```

To make them unique:

```
FD DISK1OUT  }    Would create a VALUE of ID
FD DISK2OUT  }    as DISK10 and one of DISK20,
                  thus causing no trouble during
                  object program execution.
```

Literal-1 cannot exceed six characters in length. Data-name-1 may be described as greater than six characters in length, however only the most significant six characters will be used for the "file-identifier". The literal or value of data-name-1 up to six characters is the actual value to be contained in the "file-identifier" portion of the standard magnetic tape label record or disk file header.

When data-name-1 is used the data-name must be defined in the WORKING-STORAGE section of the program and must be described as alphabetic or alphanumeric.

The VALUE OF ID declared for OUTPUT, or O-I, disk files will cause literal-1 or the value of data-name-1 up to six characters to be inserted into the disk file header. Inversely, literal-1 or the value of data-name-1 up to six characters will be checked against the MCP Disk File Directory to obtain the files physical location on disk when files are declared as being INPUT or INPUT-OUTPUT disk files.

This clause must be used if communication with specific data communication remote device is required. Literal-1 cannot exceed six characters/digits/special characters (or a mix of each) in length. The first character must be alphabetic.

The UNIT card in the MCP System Specification Deck specifies the adapter-ID to be assigned to a remote device, which will be compared with the contents of literal-1 or data-name-1 at object program execution time.

SAVE-FACTOR is not applicable to a data communication remote device.

SAVE-FACTOR is used only for output magnetic tape files. Literal-2 represents the number of days the file is to be saved before it can be manually purged and used for other purposes by the system. Literal-2 is limited to an unsigned positive integer not to exceed three digits in length with values from 001 to 999.

SAVE-FACTOR declared for a disk file is only for documentational purposes due to the fact that files residing on disk should only be purged by mutual consent within an EDP organization and can only be performed as a physical action by the systems operator.

WORK tapes are automatically assigned a SAVE FACTOR of one day to preclude expiration action when the system is being operated during the period just prior to midnight and thereafter.

RECORD DESCRIPTION.

This portion of a COBOL source program follows the file description entries and serves to completely identify each data element within a record in a given file.

The construct of these entries contain four options which are:

Option 1:

```
01 data-name-1 COPY "library-name".
```

Option 2:

```
{ 01           } { FILLER      }   [MOD] [REDEFINES data-name-2]
{ level-number } { data-name-1 }

 [ { PC      }                                       ]
 [ { PIC     }    IS    (allowable PICTURE characters) ]
 [ { PICTURE }                                       ]

 [ { BZ              } ]  [ { OC     }
 [ { BLANK WHEN ZERO } ]  [ { OCCURS }

      { integer-1 TIMES                 }
      { integer-2 TO integer-3 TIMES    }

        [DEPENDING ON data-name-3]        ]

 [ { ASCENDING  }                                            ]
 [ { DESCENDING }  KEY IS data-name-4   [data-name-5] ...    ] ...

 [ INDEXED BY index-name-1 [index-name-2] ...  ]
```

$$\left[\ [\underline{\text{USAGE}}\ \text{IS}]\ \left\{\begin{array}{l}\underline{\text{DISPLAY}}\\ \underline{\text{CMP}}\\ \underline{\text{CMP-1}}\\ \underline{\text{COMP}}\\ \underline{\text{COMPUTATIONAL}}\\ \underline{\text{COMPUTATIONAL-1}}\\ \underline{\text{COMPUTATIONAL-3}}\\ \underline{\text{INDEX}}\end{array}\right\}\ \right]\quad \left[\ \left\{\begin{array}{l}\underline{\text{JS}}\\ \underline{\text{JUST}}\\ \underline{\text{JUSTIFIED}}\end{array}\right\}\quad \underline{\text{RIGHT}}\ \right]$$

$$\left[\ \left\{\begin{array}{l}\underline{\text{VA}}\\ \underline{\text{VALUE}}\end{array}\right\}\ \text{IS literal-1}\ \left[\ \left\{\begin{array}{l}\underline{\text{THRU}}\\ \underline{\text{THROUGH}}\end{array}\right\}\ \text{literal-2}\ \right]\right.$$

$$\left.\left[\ \text{literal-3}\ \right]\ \left[\ \left\{\begin{array}{l}\underline{\text{THRU}}\\ \underline{\text{THROUGH}}\end{array}\right\}\ \text{literal-4}\ \right]\ \ldots\ \right]$$

$$\left[\ \left\{\begin{array}{l}\underline{\text{SY}}\\ \underline{\text{SYNC}}\\ \underline{\text{SYNCHRONIZED}}\end{array}\right\}\quad \left\{\begin{array}{l}\underline{\text{LEFT}}\\ \underline{\text{RIGHT}}\end{array}\right\}\ \right]\ .$$

Option 3:

$$\underline{66}\ \text{data-name-1}\ \underline{\text{RENAMES}}\ \text{data-name-2}\ \left[\ \left\{\begin{array}{l}\underline{\text{THRU}}\\ \underline{\text{THROUGH}}\end{array}\right\}\ \text{data-name-3}\ \right].$$

Option 4:

$$\underline{88}\quad \text{condition-name}\quad \left\{\begin{array}{l}\underline{\text{VA}}\\ \underline{\text{VALUE}}\end{array}\right\}\quad \text{IS literal-1}$$

$$[\text{literal-2}\ldots]\ \left[\ \left\{\begin{array}{l}\underline{\text{THRU}}\\ \underline{\text{THROUGH}}\end{array}\right\}\ \text{literal-n}\ldots\ \right].$$

The optional clauses shown may occur in any order, except if REDEFINES is used it must follow data-name-1.

The record description must be terminated by a period.

Level-numbers in Option 2 may be any number from 1-49.

The clauses PICTURE, BLANK WHEN ZERO, JUSTIFIED, and SYNCHRONIZED must occur on elementary item level only.

Option 1 can be used when the COBOL library contains the record description entry. Otherwise, one of the other options will have to be used.

In many cases, the clauses within the record description sentence are optional. Each clause is discussed in detail.

In Option 4, there is no practical limit to the number of literals in the condition-name series.

The record description for a SORTER file must be declared as 200 characters. Record positions 1-100 are for OCR and positions 101-200 are for MICR. If the record description is more than 200 characters, a syntax error will result. If the record description is less than 200 characters, the compiler will provide FILLER ADDED at the end of the description to create a record area 200 characters in length.

The record description for a LISTER file must be declared as 44 characters.

The MOD clause following the record-name will cause the beginning address of that record to start on the next modulus 1000.

BLANK WHEN ZERO.

The function of this clause is to supplement the specification of a PICTURE.

The construct of this clause is:

$$\left[\left\{\begin{array}{l}\underline{\text{BZ}}\\ \underline{\text{BLANK}}\text{ WHEN }\underline{\text{ZERO}}\end{array}\right\}\right]$$

BLANK WHEN ZERO may be abbreviated BZ.

This clause overrides the zero suppress float sign functions in a PICTURE. If the value of a field is all zeros, the BZ clause will cause the field to be edited with spaces. However, it does not override the check protect function (zero suppression with asterisks) in a PICTURE.

The BZ clause can only be used in conjunction with an item on an elementary level.

BLANK WHEN ZERO may be associated only with PICTUREs describing numeric or numeric edited fields of 99 characters or less.

CONDITION-NAME.

Condition-name is a special name which the user may assign to a given code within a data element. This value may then be referred to by the specified condition-names.

The construct of this clause is:

```
88 condition-name   { VA    }     IS literal-1
                    { VALUE }

[literal-2...]  [ { THRU    }  literal-n... ]   .
                [ { THROUGH }               ]
```

Since the testing of data is a common data processing practice, the use of conditional variables and condition-names supplies a short-hand method which enables the writer to assign meaningful names (condition-names) to particular code values that may appear in a data-field (conditional variable).

When defining condition-names, the following rules must be observed:

a. Each condition-name requires a separate entry with the level-number 88.

b. If reference to a conditional variable requires subscripting, then references to its condition-names also require subscripting.

EXAMPLES:

```
02 CONDITION-VARIABLE PC A, OCCURS 10 TIMES.
   88 GIRL    VALUE IS "G".
   88 BOY     VALUE IS "B".
   88 MAN     VALUE IS "M".
   88 WOMAN   VALUE IS "W".

IF CONDITION-VARIABLE (SUB) = "G" THEN GO TO SEE-
IF-SHES-PURDY.
```

```
IF GIRL (SUB) THEN GO TO SEE-IF-SHES-PURDY.
```

Both of the above examples will generate object code to accomplish the same result.

c. A conditional variable may be used as a qualifier for any of its condition-names.

d. Condition-names can only appear in conditional statements.

e. Condition-names cannot be associated with index-data-names.

f. Figure 4-5 provides an example of the coding of condition-name.

BURROUGHS COBOL CODING FORM

PAGE NO 1 3	PROGRAM	REQUESTED BY	PAGE OF
	PROGRAMER	DATE	IDENT 73 80

LINE NO 4 6 | 7 | A 8 11 | B 12 72

```
03 MØNTH  PICTURE 99.
    88 JANUARY  VALUE IS 01.
    88 FEBRUARY VALUE IS 02.
    88 MARCH    VALUE IS 03.
    88 APRIL    VALUE IS 04.
    88 ØTHERS   VALUE IS 05 THRU 12.
    88 ØDDS     VALUE IS 01 03 05 07 09 11.
    88 EVENS    VALUE IS 02 04 06 08 10 12.
    88 QTR-1    VALUE IS 01 THRU 03.
    88 QTR-2    VALUE IS 04 THRU 06.
    88 LASTHALF VA       IS 07 THRU 12.
    88 SELECTED VA       IS 03 THRU 05 07 09.
```

Figure 4-5. Coding of Condition-Name

DATA-NAME.

The purpose of this mandatory clause is to specify the name of each data element to be used in a program. If a data element requires a definite label, a data-name is assigned. Otherwise, the word FILLER can be used in its place.

The construct of this clause is:

$$\left\{ \begin{array}{l} \underline{\text{FILLER}} \\ \text{data-name-1} \end{array} \right\}$$

The word FILLER can be used to name a contiguous description area that does not require programmatic reference.

This entry must immediately follow a level-number. FILLER is only applicable to elementary levels.

A simple FILLER entry can, with fore-thought, cause the creation of a more efficient object code. For example, a record or data description comprised of an even number of characters may be more efficiently MOVEd than one which is comprised of an odd number of characters.

A data-name need not be unique if it can be made unique through qualification by using data-names on higher levels than itself.

It is not permissible to relationally compare an index-data-name against data-name-1.

JUSTIFIED.

The function of this clause is to specify a non-standard MOVE of alphabetic or alphanumeric data within a receiving data field.

The construct of this clause is:

$$\left[\left\{\begin{array}{l}\underline{\text{JS}}\\ \underline{\text{JUST}}\\ \underline{\text{JUSTIFIED}}\end{array}\right\} \qquad \underline{\text{RIGHT}}\right]$$

The JUSTIFIED clause can be specified only on an elementary item level where the receiving field is described as being alphabetic or alphanumeric. JUSTIFIED can be abbreviated as JS or JUST.

This clause cannot be specified for a receiving field described as being numeric or numeric edited.

When the receiving field is described with the JUSTIFIED clause and the sending field is larger than the receiving field, the left-most characters are truncated.

EXAMPLE:

SENDING		RECEIVING	
PC X(7)	A123CDE	PC X(5)	23CDE

When the receiving field is described with the JUSTIFIED clause and the sending field is smaller than the receiving field, the data will be positioned right with space fill to the left.

EXAMPLE:

SENDING		RECEIVING
PC X(5)	A123C	PC X(7)ΔΔA123C

JUSTIFIED cannot be associated with an index-data-name.

LEVEL-NUMBER.

The function of this clause is to show the hierarchy of data within a logical record. Its further function is to identify entries for condition-names, non-contiguous constants, working-storage items, and for re-grouping.

The construct of this clause is:

$$\text{level-number} \quad \left\{ \begin{array}{l} \underline{\text{FILLER}} \\ \text{data-name-1} \end{array} \right\}$$

A level-number is the first required element of each record and data-name description entry.

Level-numbers may be as follows:

a. 01 to 49 - record description and WORKING-STORAGE entries.

b. 66 - RENAMES clause used as a record description or WORKING-STORAGE entry.

c. 77 - applicable to WORKING-STORAGE only as non-contiguous items and must precede all other level numbers.

d. 88 - condition names clause used as a record description or WORKING-STORAGE entry.

Level-numbers 01 through 49 are used for record or WORKING-STORAGE descriptions. Level number 01 is reserved for the first entry within a record description. Level-number 66 is reserved for RENAMES entries. Level-number 77 is used for miscellaneous elementary items in the WORKING-STORAGE SECTION when these items are unrelated to any record. They are called non-contiguous items since it makes no difference as to the order in which they actually appear. Level-number 88 is used to define the entries relating to condition-names in record descriptions or WORKING-STORAGE entries.

For additional information on level-numbers, see LEVEL-NUMBER CONCEPT on page 4-2.

OCCURS.

The function of this clause is to define a sequence of data-items which possess identical formats, and to define a subscripted item or indices.

The construct of this clause is:

```
[ { OC     } { integer-1 TIMES                }
[ { OCCURS } { integer-2 TO integer-3 TIMES   }

     [DEPENDING ON data-name-3] ]

[ { ASCENDING  }  KEY IS data-name-4 [data-name-5]... ]  ...
[ { DESCENDING }

[    INDEXED BY index-name-1   [index-name-2] ... ]
```

This clause cannot be used in a record description entry whose level-number is 01, and can only be used with fixed-size items. Any item described with this clause must be subscripted or indexed whenever referenced in a statement other than SEARCH, and all subdivisions of the item must also be subscripted or indexed. Up to three levels of subscripting are acceptable. OCCURS can be abbreviated OC.

If only integer-1 appears, it refers to the exact number of occurrences of the data. Integer-1 must not be zero. Integer-2 TO integer-3 indicates a variable number of occurrences of this item. When integer-2 TO integer-3 is used, the following rules must be observed:

a. Integer-3 must be greater than integer-2 and both must be positive integers.

b. The item must be the last area of a record. No part of a record may follow an item of variable occurrences.

c. Only the first dimension of a table can be defined with this clause. The following definition is not permitted:

```
02 RATE-TABLE   OCCURS 10 TIMES ...
   03 WHOLE-TABLE ...
   03 AGE  OCCURS 4 TO 8 TIMES
```

d. The user must employ his own tests to determine how many occurrences of the item are actually present in the record at any time. The DEPENDING ON option is for documentational purposes only.

Integer-2 TO integer-3 indicates variable-length records and the user must specify the actual size of variable-length records in the first four bytes of each record and the record size must contain an even number of characters (MOD2). The four-character variable size indicator is counted in the physical size of each record.

The following example illustrates a use of the OCCURS clause to provide nested descriptions. A reference to ITEM-4 requires the use of three levels of subscripting; e.g., ITEM-4 (2, 5, 4). A reference to ITEM-3 requires two subscripts; e.g., ITEM-3 (I,J). In the example below there are 50 ITEM-4's.

EXAMPLE:

```
.         .         .
.         .         .
.         .         .
02 ITEM  OCCURS 2 TIMES ...
   03 ITEM-1 ...
   03 ITEM-2  OCCURS 5 TIMES ...
      04 ITEM-3 ...
      04 ITEM-4  OCCURS 5 TIMES ...
         05 ITEM-5 ...
         05 ITEM-6 ...
         .          .        .
         .          .        .
         .          .        .
```

The following example shows another use of the OCCURS clause. Assume that the user wishes to define a record consisting of five "amount" items, followed by five "tax" items. Instead of describing the record as containing 10 individual data items, it could be described in the following manner:

EXAMPLE:

```
1 TABLE ...
  2 AMOUNT  OCCURS 5 TIMES ...
  2 TAX  OCCURS 5 TIMES ...
  .        .         .
  .        .         .
  .        .         .
```

The above example would result in memory allocated for five AMOUNT fields and five TAX fields. Any reference to these fields is made by addressing the field by name (AMOUNT or TAX) followed by a subscript denoting the particular occurrence desired.

The ASCENDING/DESCENDING KEY option is for documentation only.

The operands in the INDEXED BY option are index-names or indices. The operands of an INDEXED BY option must appear in association with an OCCURS clause and are usable only when referencing that level of the table. When using three-level indexing, each level must have an INDEXED BY option and in a given indexing operation, only one operand from each option may be used.

Other than their use as an index into an array, an index-name may be referred to only in a SET, SEARCH, PERFORM, or in a relation condition. All index-names must be unique. Index-names have an assumed construction of PC 9(5) COMPUTATIONAL.

Using an index-name associated with one (row of a) table for indexing into another (row of a) table will not cause a syntax error, but will, in most cases, cause incorrect object time results since it is the index-name that contains the information pertinent to the element sizes.

When using an index-name series (e.g., INDEXED BY A, B, C):

a. The indexes should be used only when referencing the associated row.

b. All "assumed" reference are to the first index-name in a series. Others in the series are affected only during an explicit reference.

Indexing into a table follows much the same logic as subscripting. There is a limit of three indexes per operand (e.g., A (INDEX-1, INDEX-2, INDEX-3). The use of a relative index allows modification of the index-name without actually changing the value of the index-name.

EXAMPLE:

```
A (INDEX-1 +3, INDEX-2 -4, INDEX-3)
```

Relative indexing is indicated by a + or a - integer following an index-name and causes the affected index to be incremented or decremented by that number of elements within the table.

A data-name whose USAGE is defined to be INDEX is an index-data-name.

Condition-names, PICTURE, VALUE, SYNCHRONIZED or JUSTIFIED <u>cannot</u> be associated with an index-data-name.

The B 2500/B 3500 COBOL Compilers will assign the construction of a PC 9(5) COMPUTATIONAL area for each index-data-name specified.

It is not permissible to relationally compare an index-data-name against a literal or a regular data-name.

PICTURE.

The function of this clause is to describe the size, class, general characteristics, and editing requirements of an elementary item.

The construct of this clause is:

$$\left\{\begin{array}{l}\underline{\text{PC}}\\ \underline{\text{PIC}}\\ \underline{\text{PICTURE}}\end{array}\right\}\qquad \text{IS}\quad \text{(character string)}$$

The word PICTURE may be abbreviated as PC or PIC. Character string denotes letters of the alphabet, special characters, and digits which are used in conjunction with one another to describe a data-name. See USAGE for a description of characters and digits.

The maximum number of characters and symbols allowed in the character string used to describe a data-name or FILLER, is 30. A character string consists of a certain allowable combination of characters defined as PICTURE descriptors, plus insert characters encompassing the entire character set employed by the systems line printer that have no PICTURE descriptor value or action.

This clause must appear for every elementary item level entry and cannot be used at group levels.

PICTURE cannot be associated with an index-data-name.

A PICTURE of A(5) indicates that the item is a five character (byte) alphabetic field. The integer within parentheses indicates how many times A occurs in order to constitute the desired PICTURE. The PICTURE A(5) can also be represented by AAAAA. The value of the integer within parentheses must always be greater than zero.

Record descriptions do not necessarily have to conform to the physical characteristics of an ASSIGNed hardware-name. The flow of input-output data will terminate at the end of the prescribed PICTURE size. For example:

READER (can read 80 columns) description can be PICTUREd from 1 through 80.

PUNCH (can punch 80 columns) description can be PICTUREd from 1 through 80.

PRINTER (120/132 character lines) description can be PICTUREd from 1 through maximum.

SPO (one character at a time) description can be PICTUREd from 1 to any limit.

There are five categories of data that can be described with a PICTURE clause. These are alphabetic, numeric, alphanumeric, alphanumeric-edited, and numeric-edited.

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

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

b. The letter B in a character string represents a position into which the space character is to be inserted.

c. The letter J in a character string indicates that the operational data sign is appearing as an over-punch in the least-significant digit position if USAGE IS DISPLAY is associated with the item. However, if USAGE has been indicated as COMPUTATIONAL, J takes on the same function as an S. A J is not counted in the length of a DISPLAY item. Only one operational sign may appear in any one PICTURE and, if specified, the J must appear as the left-most character of the PICTURE. Data elements requiring a J PICTURE descriptor may not be described by a VALUE clause with a signed literal. PICTURE J should be used only in those cases where PICTURE S is not applicable.

NOTE

If J appears within a PICTURE descriptor, it no longer performs as an operational sign but serves to reinitiate zero suppression.

d. The letter K in a character string indicates the presence of an 8-bit (byte) sign appearing in the first character position of a PICTURE descriptor when USAGE is implicitly or explicitly DISPLAY and is counted in the length of the PICTURE. If USAGE IS COMPUTATIONAL, the letter K becomes the same as an S. Data elements requiring a K PICTURE descriptor may not be described by a VALUE clause with a signed literal.

e. The letter P in a character string indicates an assumed decimal scaling position and is used for specifying 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 length of the allowable number of characters within a PICTURE description. Scaling position characters are counted in determining the maximum number of digit positions (99) in numeric edited items or numeric items which appear as operands in arithmetic statements. The character P can appear only to the left or right as a continuous string of P's within a PICTURE description. Since it implies an assumed decimal point (to the left of the P's if P's are left-most PICTURE characters and to the right of P's if P's are right-most PICTURE characters), the assumed decimal point symbol V is redundant as either the left-most of right-most character within such a PICTURE description.

f. The letter S in a character string is used to indicate the presence of the standard operational sign in the form of an overpunch in the most-significant digit position of an item if USAGE IS DISPLAY and is not counted in the length of the PICTURE. If USAGE IS CMP, it will denote an operational sign digit in front of the most-significant digit position and is counted in the length of the PICTURE. The S must be written as the first character of the character string of a PICTURE. A signed item may not be more than 99 characters/ digits in length. Wherever possible, PICTURE S should be used rather than J or K.

g. The letter V in a character string indicates the location of an assumed decimal point and may only appear once in a character string. It does not represent a character position and therefore is not counted in the length of the item. When the assumed decimal point character V is the right-most character of the PICTURE character string, it is redundant. The maximum number of decimal places is 99.

h. The letter X in a character string indicates an alphanumeric position which can contain any allowable character in the computer's character set.

i. Each letter Z in a character string represents a zero suppress editing action and may only be used to cause the left-most leading numeric character positions to be replaced by a space at object time when the contents of that character position is zero. Each Z is counted as part of the PICTURE length. Zero suppression is terminated with the first non-zero numeric character in the data. Insertion characters are also replaced by spaces while suppression is in effect. Z can also appear to the right of J when the J symbol is used to reinitiate zero suppression. For additional information on zero suppression, see the BLANK WHEN ZERO clause. FILLER entries cannot be defined by the letter Z usage.

j. The number 9 in a character string represents numeric data. If USAGE IS explicitly or implicitly DISPLAY, the data will be operated on as 8-bit (byte) characters. If USAGE IS CMP, it will be operated on as 4-bit digits. Each 9 is counted in the length of the PICTURE.

k. The number 0 (zero) in a character string represents a position into which zero is to be inserted when that item is a receiving field and it is counted in the length of the PICTURE.

l. The special character comma in a character string represents a position into which a comma will have to be inserted. It is counted as part of the PICTURE length. (Also see DECIMAL-POINT IS COMMA in section 3 of this document.) If zero suppression is indicated, a blank character will replace each applicable comma until meaningful data is encountered in the data stream.

m. The special character period in a character string is an editing symbol which represents the decimal point for data alignment purposes. In addition, it represents a character position into which a period will be inserted. It is counted as part of the PICTURE length. If more than one period is indicated in the PICTURE, the left-most period determines the scale of the PICTURE. The PICTURE must not terminate with a period except when it is used to indicate the end of the item clause. For a given program, the function of the period and comma are exchanged if the DECIMAL-POINT IS COMMA clause appears in the SPECIAL-NAMES paragraph. If exchanged, the rules that apply to the use of periods apply to commas and vice versa. (Also see DECIMAL-POINT IS COMMA in section 3 of this document.)

n. The symbols +, -, CR, and DB are used as editing sign control symbols. When used, they represent the character position into which the editing sign control symbol will be placed. The symbols are mutually exclusive in any one character string and each character used in the PICTURE is counted in the length.

1) Fixed insertion characters. A single + or - can be used at the extreme left or right of a PICTURE. The CR and DB can be used only at the extreme right end of a PICTURE. The CR and DB symbols represent a two character position and are counted in the length of the item. Only one currency symbol and only one of the editing sign control symbols can be used in a given PICTURE. The currency symbol ($) must be the leftmost character position except that it can be preceded by either a + or - symbol. Fixed insertion editing results in the insertion character occupying the same character position in the edited item as it occupied in the PICTURE character string. Editing sign control symbols (sometimes referred to as report signs) produce the results shown in Table 4-3, depending upon the value that the item contains.

Table 4-3
Editing Sign Control Symbol Results

Editing Symbol In Picture Character String	Result	
	Data Item Positive	Data Item Negative
+	+	-
-	space	-
CR	2 spaces	CR
DB	2 spaces	DB

2) Floating Insertion Characters. When used as floating replacement and suppression characters, + and - are written from the extreme left of the PICTURE to

represent each leading numeric character into which the sign (+ or -) is to be floated. At least two symbols must be shown to use the subject symbols as floating characters. The floating symbol may not appear to the right of the decimal point unless all replacement positions consist of that symbol. In this case, the field will consist of all spaces when the value is zero. The currency symbol and editing symbols + and - are the insertion characters, and they are mutually exclusive as floating insertion characters in a PICTURE character string.

3) In a PICTURE character string, there are only two ways of representing floating insertion editing. One way is to represent, by the insertion characters, any or all of the leading numeric character positions to the left of the decimal point. The other way is to represent all of the numeric character positions in the PICTURE character string by the insertion characters. If the first method is employed, a single insertion character will be placed into the character position immediately preceding the first non-zero digit in the data represented by the insertion symbol string to the decimal point, whichever is encountered first. If the second method is used, the result depends upon the value of the data. If the value is zero, the entire data item will contain spaces. If the value is not zero, the result is the same as when the insertion character is only to the left of the decimal point. The PICTURE must contain at least one more floating insertion character than the maximum number of significant digits in the item to be edited.

o. The special character asterisk in a character string represents a leading numeric character position into which an asterisk will be placed when the content of that position is zero and asterisk replacement has not disabled. Asterisk replacement is disabled when the first non-zero character is encountered, or when the decimal point (implicit or explicit) is reached. When the PICTURE character string specifies only asterisks (*), and the value of the item is zero, the entire output item will consist of asterisks and the decimal point, if present. BLANK WHEN ZERO does not override the insertion of asterisks.

p. The special character dollar sign in a character string represents a character position into which a currency symbol is to be inserted. The currency symbol in a character string is represented automatically by a dollar sign ($). If the CURRENCY clause of the SPECIAL-NAMES paragraph is indicated, the dollar sign is replaced by the character specified as a replacement CURRENCY SIGN and is counted in the length of the item.

1) Fixed insertion character. The currency sign may appear anywhere in the PICTURE.

2) Floating insertion character. At least two currency signs must appear as the left-most characters in the PICTURE. The currency sign is written to represent each leading numeric character position into which the currency sign may be floated. A single sign is placed in the least-significant suppressed position shown by the currency symbol in the PICTURE. The output item must contain at least one more currency sign character position than the maximum number of significant digits in the source item.

The length of an elementary item, where the length means the number of character positions occupied by the elementary item in standard data format, is determined by the number of allowable symbols which represent character positions.

An integer which is enclosed in parentheses describing the character string of a PICTURE and following the symbols A, ', X, 9, P, Z, *, B, 0, +, -, or the currency sign indicates the number of consecutive occurrences of the symbol. Note that the K, S, CR, and DB symbols may appear only once in a given PICTURE character string.

To define an item as alphabetic, its PICTURE character string can only contain the symbols A and B.

To define an item as numeric, its character string of the PICTURE can only contain the symbols 0, 9, J, K, P, S, and V. Its contents, when represented in standard data format, must be a combination of the numerals 0, 1 through 9. The item may include an operational sign symbol.

To define an item as alphanumeric, its PICTURE character string is restricted to certain combinations of the symbols A, X, and 9.

The item is treated as if the character string contained all X's. The PICTURE character string which contains all A's or all 9's does not define an alphanumeric item.

To define an item as alphanumeric edited, its PICTURE character string is restricted to the following combinations of symbols:

a. The character string must contain at least one B, one X, and one 0 (zero).

b. Another alternative is that the character string must have at least one 0 (zero) and one A.

To define an item as numeric edited, its PICTURE character string is restricted to certain combinations of the symbols B, J, K, P, V, Z, 0, 9, comma, period, *, +, -, CR, DB, and the currency sign. The allowable combinations are determined by the order of precedence of symbols and the editing rules. The number of positions which may be represented in the character string is 99.

There are two general methods of performing editing in the PICTURE clause, either by insertion or by suppression and replacement. There are four types of insertion editing available.

a. Simple insertion.
b. Special insertion.
c. Fixed insertion.
d. Floating insertion.

There are two types of suppression and replacement editing modes:

a. Zero suppression and replacement with spaces.
b. Zero suppression and replacement with asterisks.

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

Simple insertion editing involves the usage of comma, B, and 0 (zero) as the insertion characters. The insertion characters are counted in the length of the item and represent the position in the item into which the character will be inserted.

Special insertion editing character period (.) is used to represent the decimal point for alignment in addition to acting as an insertion character. The insertion character used for the actual decimal point is counted in the length of the item. The use of the assumed decimal point, represented by the symbol V and the actual decimal point, represented by the insertion character period (.) in the same PICTURE character string is disallowed. If the insertion character is the last symbol in the character string, it must be immediately followed by one of the punctuation characters, semicolon, or period, followed by a space. The result of special insertion editing is the appearance of the insertion character in the item in the same position as shown in the character string. Any character or digit other than those defined with PICTURE meanings can be used as special insertion characters and will be counted in the size of the PICTURE.

EXAMPLE:

> 99/99/99 could be a date mask and 999=99=9999 could represent a social security number mask.

Zero suppression editing of leading zeros in numeric character positions is indicated by the use of the character Z, or the character * (asterisk) as suppression symbols in a PICTURE character string. These symbols are mutually exclusive in a given PICTURE character string. Each suppression symbol is counted in determining the length of the item. If Z is used, the replacement character will be the space and if the asterisk is used, the replacement character will be *.

Zero suppression and replacement is indicated in a PICTURE character string by using a string of one or more of the allowable symbols to represent leading numeric character positions which are to be replaced when the character contains a zero. Any of the simple insertion characters embedded in the string of symbols or to the immediate right of this string are part of the string.

In a PICTURE character string, there are two ways of representing zero suppression. One way is to represent any or all of the leading numeric character positions to the left of the decimal point by suppression characters. The other way is to represent all of the numeric character positions in the PICTURE character string by suppression characters. If the suppression symbols appear only to the left of the decimal point, any leading zero in the data which corresponds to a symbol in the string is replaced by the replacement character. Suppression terminates at the first non-zero digit in the data represented by the suppression symbol string or at the decimal point, whichever is encountered first. If all numeric positions in the PICTURE character string are represented by suppression symbols, and the value of the data is not zero, the result is the same as if the suppression characters were only to the left of the decimal point. If the value is zero, the entire data item will be spaces if the suppression symbol is Z, whereas asterisks will cause the field (except for decimal point) to be replaced with asterisks. Even if the BLANK WHEN ZERO clause is used in conjunction with asterisks, the replacement of character positions containing zeros will be conducted with asterisks.

The symbols +, -, *, Z, and the currency symbol, when used as floating replacement characters, are mutually exclusive within a given character string. At least two floating replacement characters must appear as the left-most characters in the PICTURE.

Table 4-4 shows the order of precedence when using characters as symbols in a character string. For a given function in the left column, a small x in its row indicates that the arguments, used as column headings, are the only ones that may immediately precede the first appearance of the function in a particular string. Arguments appearing in braces ({}) indicate that the symbols are mutually exclusive. The currency symbol is represented by $.

Table 4-4

Order of Precedence When Using Characters As Symbols

	S	P	$	{+	-}	{ZZ	**	$$	++	--}
S										
P	x			(1)	(1)	x	x	x	x	x
$	x	x		x	x					
+		x				x	x	x		
-		x				x	x	x		
ZZ		x	x	x	x					
**		x	x	x	x					
$$		x		x	x					
++		x	x							
--		x	x							

The symbols A, B, V, X, 0, 9, period, and comma can be preceded by any symbols in the PICTURE character string except CR and DB.

NOTE

When the + or - appears on the right of a character string and the P is also on the right, P precedes the sign indicator.

To simplify the explanation of allowable character pairs in the character string of a PICTURE, table 4-5 and 4-6 are provided. These tables have been constructed so that they reflect the use of all allowable symbols, depending upon whether the item is numeric, alphabetic, or alphanumeric. For example, if the item is numeric and the programmer wishes to determine whether the symbol V can follow a 9, then table 4-5 should be used. In the numeric item section of table 4-5, the letter Y (Yes) can be found at the crossing point of horizontal, first symbol, 9 and vertical, second symbol, V. On the other hand, the use of J after 9 is indicated with N (No).

Table 4-5

Numeric or Alphabetic Items

			SECOND SYMBOL							
			Numeric Item						Alphabetic Item	
			9	V	S	J	K	P	A	B
FIRST SYMBOL	Numeric Item	9	Y	Y	N	N	N	Y		
		V	Y	N	N	N	N	Y		
		S	Y	Y	N	N	N	Y		
		J	Y	Y	N	N	N	Y		
		K	Y	Y	N	N	N	Y		
		P	Y	Y	N	N	N	Y		
	Alphabetic Item	A							Y	Y
		B							Y	Y

Table 4-6

Alphanumeric Items

		SECOND SYMBOL																	
		Non-Editing				Editing													
FIRST SYMBOL		9	X	A	B	J	9	V	,	.	+	-	Z	*	CR	DB	B	O	$
NoN Editing	9	Y	Y	Y	Y														
	X	Y	Y	Y	Y														
	A	Y	Y	Y	Y														
	B	Y	Y	Y	Y														
Editing	9					Y	Y	Y	Y	Y	Y	Y	N	N	Y	Y	Y	Y	N
	V					Y	Y	N	N	N	Y	Y	Y	Y	Y	Y	Y	Y	N
	,					Y	Y	N	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	N
	.					Y	Y	N	Y	N	Y	Y	Y	Y	Y	Y	Y	Y	N
	+					Y	Y	Y	Y	Y	Y	N	Y	Y	N	N	N	Y	Y
	-					Y	Y	Y	Y	Y	N	Y	Y	Y	N	N	N	Y	Y
	Z					Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	N
	*					Y	Y	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	Y	N
	CR					N	N	N	N	N	N	N	N	N	N	N	N	N	N
	DB					N	N	N	N	N	N	N	N	N	N	N	N	N	N
	B					Y	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	N
	O					Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	N
	$	(BUT NOT FIRST SYMBOL IN PC)				Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	J					Y	Y	Y	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	Y

Table 4-7 demonstrates the editing function of the PICTURE clause.

Table 4-7

Editing Application of the Picture Clause

Source Area		Receiving Area	
Picture	Data	Editing Picture	Edited Data
9(5)	12345	$ZZ,ZZ9.99	$12,345.00
V9(5)	12345	$$$,$$9.99	$0.12
V9(5)	12345	$ZZ,ZZ9.99	$ 0.12
9(5)	00000	$$$,$$9.99	$0.00
9(3)V99	12345	$ZZ,ZZ9.99	$ 123.45
9(5)	00000	$$$,$$$.$$	
9(5)	01234	$$$,$$$.$$	$ 1,234.00
9(5)	00000	$**,**9.99	*******.**
9(5)	00123	$**,***.**	$***123.00
9(3)V99	00012	$ZZ,ZZ9.99	$ 0.12
9(3)V99	12345	$$$,$$9.99	$123.45
9(3)V99	00001	$ZZ,ZZZ.99	$ .01
9(5)	12345	$$$,$$9.99	$12,345.00
9(5)	00000	$ZZ,ZZZ.ZZ	
9(3)V99	00001	$$$,$$$.$$	$.01
S9(5)	(+) 12345	ZZZZ9.99+	12345.00+
S9(5)	(-) 00123	--99999.99	- 123.00
9(3)V99	12345	999.00	123.00
S9(5)	(-) 12345	ZZZZ9.99-	12345.00-
S9(5)	(+) 12345	ZZZZ9.99-	12345.00
9(5)	12345	BBB99.99	45.00
S9(5)V	(-) 12345	-ZZZZ9.99	-12345.00
S9(5)	(-) 12345	$$$$$$.99CR	$12345.00CR
S99V9(3)	(-) 12345	------.99	-12.34
S9(5)	(+) 12345	$$$$$$.99CR	$12345.00
9(3)V99	12345	999.BB	123.
9(5)	12345	00999.00	00345.00
9(7)	0012003	ZZ99JZ9	12 3

REDEFINES.

The function of this clause is to allow an area of memory to be referred to by more than one data-name with different formats and sizes.

The construct of this clause is:

```
[level-number data-name-1 REDEFINES data-name-2]
```

The level-numbers of data-name-1 and data-name-2 must be identical and must not be 66 or 88.

This clause must not be used in 01 level entries of the FILE SECTION as an implicit REDEFINES is assumed when multiple 01 level entries within a file description are present. The size of the record(s) causing implicit redefinition do not have to be equal to that of the record being redefined. The various sizes of implicitly redefined record descriptions create no restriction as to which description is to be coded first, second, third, etc., in the source program.

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

When the level-number of data-name-2 is other than 01 (REDEFINES can not be used on the 01 level in the FILE SECTION), it must specify a storage area of the same size as specified by data-name-1. It is important to observe that the REDEFINES clause specifies the redefinition of a storage area, not simply of the data items occupying that area.

The entries giving the new description of the storage area must immediately follow the entries describing the area being redefined.

The data description entry being redefined cannot contain an OCCURS clause, nor can it be subordinate to an entry which contains an OCCURS clause.

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

Data-name-2 need not be qualified.

RENAMES.

The function of this clause is to permit alternative and possibly overlapping, grouping of elementary items.

The construct of this clause is:

$$66\ \text{data-name-1}\ \underline{\text{RENAMES}}\ \text{data-name-2}\ \left[\left\{\begin{matrix}\underline{\text{THRU}}\\ \underline{\text{THROUGH}}\end{matrix}\right\}\ \text{data-name-3}\right].$$

All RENAMES entries associated with a given logical record must immediately follow its last data description entry.

Data-name-2 and data-name-3 must be names in the associated logical record and cannot be the same data-name or have the same logical address. A 66 level entry cannot rename another 66 level entry nor can it rename a 77, 88, or 01 level entry.

Data-name-1 cannot be used as a qualifier, and can only be qualified by the names of the level 01 or FD entries. Neither data-name-2 nor data-name-3 may have an OCCURS clause in its data description entry nor be subordinate to an item that has an OCCURS clause in its data description entry.

Data-name-2 must precede data-name-3 in the Record Description, and data-name-3 cannot be subordinate to data-name-2.

One or more RENAMES entries can be written for a logical record.

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

When data-name-3 is not specified, data-name-2 can be either a group or an elementary item. When data-name-2 is a group item, data-name-1 is treated as a group item, and when data-name-2 is an elementary item, data-name-1 is treated as an elementary item.

When data-name-3 is specified, none of the elementary items within the range, including data-name-2 and data-name-3, can be of variable length.

SYNCHRONIZED.

The function of this clause is to specify positioning of an elementary item within a computer word or words.

The construct of this clause is:

$$\left[\begin{Bmatrix}\underline{\text{SY}}\\ \underline{\text{SYNC}}\\ \underline{\text{SYNCHRONIZED}}\end{Bmatrix}\quad\begin{Bmatrix}\underline{\text{LEFT}}\\ \underline{\text{RIGHT}}\end{Bmatrix}\right]$$

SYNCHRONIZED may be abbreviated as SY or SYNC and may only appear with a description of an elementary item.

This clause specifies that the compiler, in creating the internal format of this item, must place the item in the minimum number of computer words which can contain the item. If the size of the item, explicitly or implicitly, is not an exact multiple of the number of characters in a computer word, the character positions between the item and the computer word boundary cannot be assigned to another item. Such unused character positions are included in:

a. The length of any group to which the elementary item belongs.

b. The computer storage area allocation when the elementary item appears as the object of a REDEFINES clause.

SYNCHRONIZED LEFT specifies that the elementary item is to be positioned so that it will begin at the left boundary of a computer word.

SYNCHRONIZED RIGHT specifies that the elementary item be positioned so that it will terminate at the right boundary of a computer word.

Whenever the SYNCHRONIZED item is referenced in the source program, the original length of item, as shown in the PICTURE clause, is used in determining any action which depends on the length, such as justification, truncation, or overflow.

If the data description of an item contains the SYNCHRONIZED clause and an operational sign, the sign of the item appears in the normal operational sign position regardless of whether the item is SYNCHRONIZED RIGHT or SYNCHRONIZED LEFT.

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

A word in the B 2500/B 3500 contains two 8-bit characters (2 bytes).

SYNCHRONIZED cannot be associated with an index-data-name.

USAGE.

The function of this clause is to specify the format of a data item in compiler storage.

The construct of this clause is:

```
                       ⎧ DISPLAY            ⎫
 ⎡                     ⎪ CMP                ⎪ ⎤
 ⎢                     ⎪ CMP-1              ⎪ ⎥
 ⎢  [USAGE IS]         ⎨ COMP               ⎬ ⎥
 ⎢                     ⎪ COMPUTATIONAL      ⎪ ⎥
 ⎢                     ⎪ COMPUTATIONAL-1    ⎪ ⎥
 ⎣                     ⎪ COMPUTATIONAL-3    ⎪ ⎦
                       ⎩ INDEX              ⎭
```

The USAGE clause can be written at any level. If USAGE is written on group level, it applies to each elementary item in that group.

COMPUTATIONAL-1, CMP-1, or COMPUTATIONAL-3 are acceptable substitutes for, and are equivalent to, COMPUTATIONAL, COMP, or CMP entries.

A warning message of POSSIBLE CMP GROUP USAGE ERROR will appear whenever the receiving field is a group CMP item. It indicates that the resultant contents during object program execution of the group CMP item may not contain expected results.

NOTE

> Group moves are performed whenever the sending <u>or</u> receiving field is a group item and both will be treated as alphanumeric (byte) data, regardless of USAGE.

The USAGE of an elementary item cannot contradict the USAGE of a group to which the item belongs.

USAGE is a declaration for the EBCDIC internal representation of the system and is defined as follows:

a. When USAGE IS DISPLAY, the data item consists of 8-bit (byte) characters; two such characters comprise a B 2500/ B 3500 computer word.

b. When USAGE IS COMPUTATIONAL, the data item consists of 4-bit coded digits.

c. When USAGE IS INDEX, a PICTURE may not be specified.

The PICTURE of a COMPUTATIONAL item can contain only 9's, the operational sign character S, J, or K, the decimal point character V, one or more P's and the insertion character 0 (zero).

COMPUTATIONAL items may be declared for 9-channel magnetic tape files (TAPE-9), disk file (DISK), Supervisory Printer, paper tape files (PT-READER or PT-PUNCH), or for WORKING-STORAGE SECTION items.

A DISPLAY item is automatically converted to its 4-bit equivalent whenever the receiving area is defined as COMPUTATIONAL except when the receiving area is a group item. A CMP item is automatically converted to its 8-bit equivalent whenever the receiving area is declared DISPLAY except when the sending CMP item is a group item.

Arithmetic operations utilizing COMPUTATIONAL and DISPLAY operands in the same statement arc of no concern to the user. The B 2500/ B 3500 efficiently uses these operands with no prior conversion of data format by the programmer.

In the absence of a USAGE clause, USAGE IS DISPLAY will be assumed.

For the most efficient use of hardware storage and internal record storage areas, records should be devised so as to avoid inter-mixing of odd-length COMPUTATIONAL items with DISPLAY items. This rule is due to the compiler automatically placing the machine addresses of DISPLAY areas to modulo two. For example:

BAD RECORD LAYOUT

```
03  data-name-1  PC 9 USAGE IS DISPLAY }
                                       } (takes up one word)
03  data-name-2  PC 9 USAGE IS CMP.    }
    FILLER ADDED (see note below) .
03  data-name-3  PC 9 USAGE IS DISPLAY. }
                                        } (depends on next entry)
03  data-name-4  PC 9 USAGE IS CMP.     }
03  data-name-n...
 .
 .
 .
```

NOTE

FILLER PC 9 USAGE IS CMP was automatically inserted to move the location counter to MOD 2.

GOOD RECORD LAYOUT (Rearrangement of the above record)

```
03  data-name-1  PC 9 USAGE IS DISPLAY. }
                                        } (takes up one word)
03  data-name-3  PC 9 USAGE IS DISPLAY. }
03  data-name-2  PC 9 USAGE IS CMP.    }  (takes up one-
03  data-name-4  PC 9 USAGE IS CMP.    }   half word)
03  data-name-n  ...                      (depends on next entry)
```

The compiler adjusts the resultant object code addresses of group items to byte boundaries. The following examples reflect four COMPUTATIONAL data fields in two different data element arrangements:

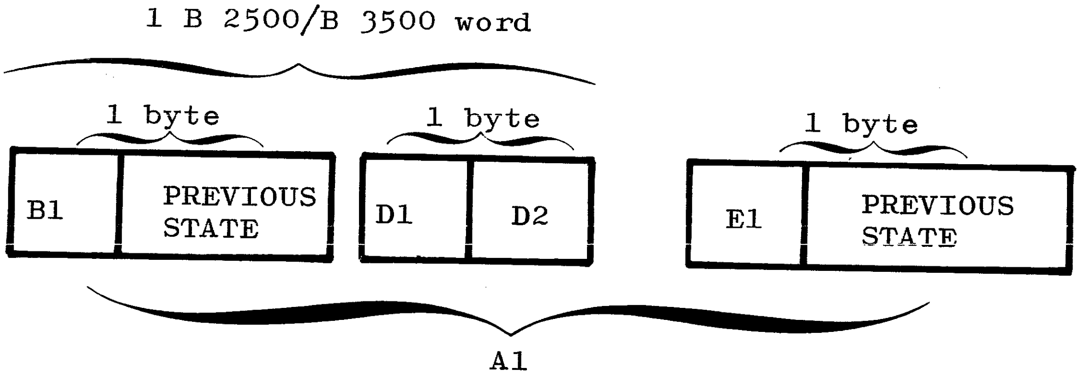
EXAMPLE 1

```
01 A.
   02 B   PC 9 CMP.
   02 C   PC 9 CMP.
   02 D   PC 9 CMP.
   02 E   PC 9 CMP.
   02 F. . .
```

EXAMPLE 2

```
01 A1.                     (group)
      02 B1 PC 9 CMP.      (elem.)
      02 C1.               (group)
           03 D1  PC 9 CMP. (elem.)
           03 D2  PC 9 CMP. (elem.)
      02 E1 PC 9 CMP.      (elem.)
      02 F1.               (group)
```

Record A will occupy two contiguous bytes of memory, while record A1 will consist of three contiguous bytes. The A1 description will cause the following object code format to be produced:



Data-names whose USAGE IS INDEX are referred to as index-data-names. They are never referred to as an "index", cannot be used for indexing and are not associated with a table. They should be considered only as temporary storage areas for index-names. They may be referenced only in a SET or SEARCH statement or in a relation condition.

Condition-names, PICTURES, VALUE, SYNCHRONIZED or JUSTIFIED cannot be associated with an index-data-name.

Every index-data-name will be automatically assigned a PC 9(5) COMPUTATIONAL area by the B 2500/B 3500 COBOL Compilers.

VALUE.

The function of this clause is to declare an initial value to WORKING-STORAGE items, or the value associated with a condition-name.

The construct of this clause is:

$$\left[\begin{Bmatrix}\underline{\text{VA}}\\ \underline{\text{VALUE}}\end{Bmatrix}\right. \text{ IS literal-1 } \left[\begin{Bmatrix}\text{THRU}\\ \underline{\text{THROUGH}}\end{Bmatrix}\text{ literal-2}\right] \left[\text{literal-3}\right]$$

$$\left[\begin{Bmatrix}\text{THRU}\\ \underline{\text{THROUGH}}\end{Bmatrix}\text{ literal-4}\right] \left.\ldots\right]$$

Abbreviation VA can be used in lieu of VALUE.

Literals may consist of Figurative Constants; e.g., ZEROS, QUOTES, etc.

Literals may be replaced by the reserved word DATE-COMPILED. If DATE-COMPILED is used in the VALUE clause, the date that the program was compiled will be placed in the data-name in the JULIAN form of YYDDD.

In the FILE SECTION, the VALUE clause is allowed only in condition-name (88 level) entries. VALUE entries in other data descriptions in the FILE SECTION are considered as being documentation only.

In the WORKING-STORAGE SECTION, the entire VALUE clause may be used with condition-name entries. All levels other than 88 are restricted to the use of literal-1 only.

The VALUE clause must not be stated in a Record Description entry with an OCCURS clause, or in an entry which is subordinate to an entry containing an OCCURS clause. This rule does not apply to condition-name entries.

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

a. If the category of an item is numeric, all literals in the VALUE clause must be numeric literals; e.g., VA 1, 3 THRU 9, 12, 16 THRU 20, 25 THRU 50, 51, 56.

b. If the category of the item is alphabetic or alphanumeric, all literals in the VALUE clause must be specifically stated non-numeric literals; e.g., VA IS "A", "B", "C", "F", "M", "N", "O", "P", "Q", "Z".

c. All literals in a VALUE clause of an item must have a value which requires no editing to place that value in the item as indicated by the PICTURE clause.

d. The function of any editing clauses or editing characters in a PICTURE clause is ignored in determining the initial appearance of the item described. However, editing characters are included in determining the length of the item.

In a condition-name entry, the VALUE clause is required and is the only clause permitted in the entry. The characteristics of a condition-name are implicitly those of its conditional variable.

If this clause is used in an entry at the group level, the literal must be a figurative constant or a non-numeric literal (byte characters). The group area is initialized without consideration for the USAGE of the individual elementary items. Subordinate levels within the group cannot contain VALUE clauses.

The VALUE clause must not be specified for a group containing items requiring separate handling due to the SYNCHRONIZED or USAGE clause.

The VALUE clause must not be stated in a Record Description entry which contains a REDEFINES clause, or in an entry which is subordinate to an entry containing a REDEFINES clause. This rule does not apply to condition-name entries.

A literal must not contain a sign when the VALUE clause is used with a data-name whose PICTURE specifies a J or K sign position.

In a VALUE clause, there is no practical limit to the number of literals in a series. VALUE cannot be associated with an index-data-name.

WORKING-STORAGE SECTION.

The WORKING-STORAGE SECTION is optional and is that part of the DATA DIVISION set aside for intermediate processing of data. The difference between WORKING-STORAGE and the FILE SECTION is that the former deals with data that is not associated with an input or output file.

ORGANIZATION.

Whereas the FILE SECTION is composed of file description (FD or SD) entries and their associated record description entries, the WORKING-STORAGE SECTION is composed only of record description entries and non-contiguous items. The WORKING-STORAGE SECTION begins with a section-header and a period, followed by item description entries for non-contiguous WORKING-STORAGE items, and then by record description entries for WORKING-STORAGE records, in that order. The format for WORKING-STORAGE SECTION is as follows:

```
[WORKING-STORAGE SECTION]
    [77 data-name-1]
        [88 condition-name-1]
                .
                .
    [77 data-name-n]
    [01 data-name-2]
        [02 data-name-3]
                .
                .
                .
    [66 data-name-m RENAMES data-name-3]
    [01 data-name-4]
        [02 data-name-5]
            [03 data-name-n]
            [88 condition-name-2]
```

NON-CONTIGUOUS WORKING-STORAGE.

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:

a. Level-number.

b. Data-name.

c. PICTURE clause or equivalent.

The OCCURS clause is not meaningful on a 77 level item and will cause 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.

All level 77 items must appear before any 01 levels in WORKING-STORAGE.

WORKING-STORAGE RECORDS.

Data elements in WORKING-STORAGE which bear a definite relationship to one another must be grouped into records according to the rules for the 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, OCCURS, and COPY. Each WORKING-STORAGE record-name (01 level) must be unique since it cannot be qualified by a file-name. Subordinate data-names need not be unique if they can be made unique by qualification.

INITIAL VALUES.

The initial value of any item in the WORKING-STORAGE SECTION is specified by using the VALUE clause of the record description. If VALUE is not specified, the initial values are set to 4-bit zeros (COMPUTATIONAL).

CONDITION-NAMES.

Any WORKING-STORAGE item may be a conditional variable with which one or more condition-names are associated. Entries defining condition-names must immediately follow the conditional variable entry. Both the conditional variable entry and the associated condition-name entries may contain VALUE clauses.

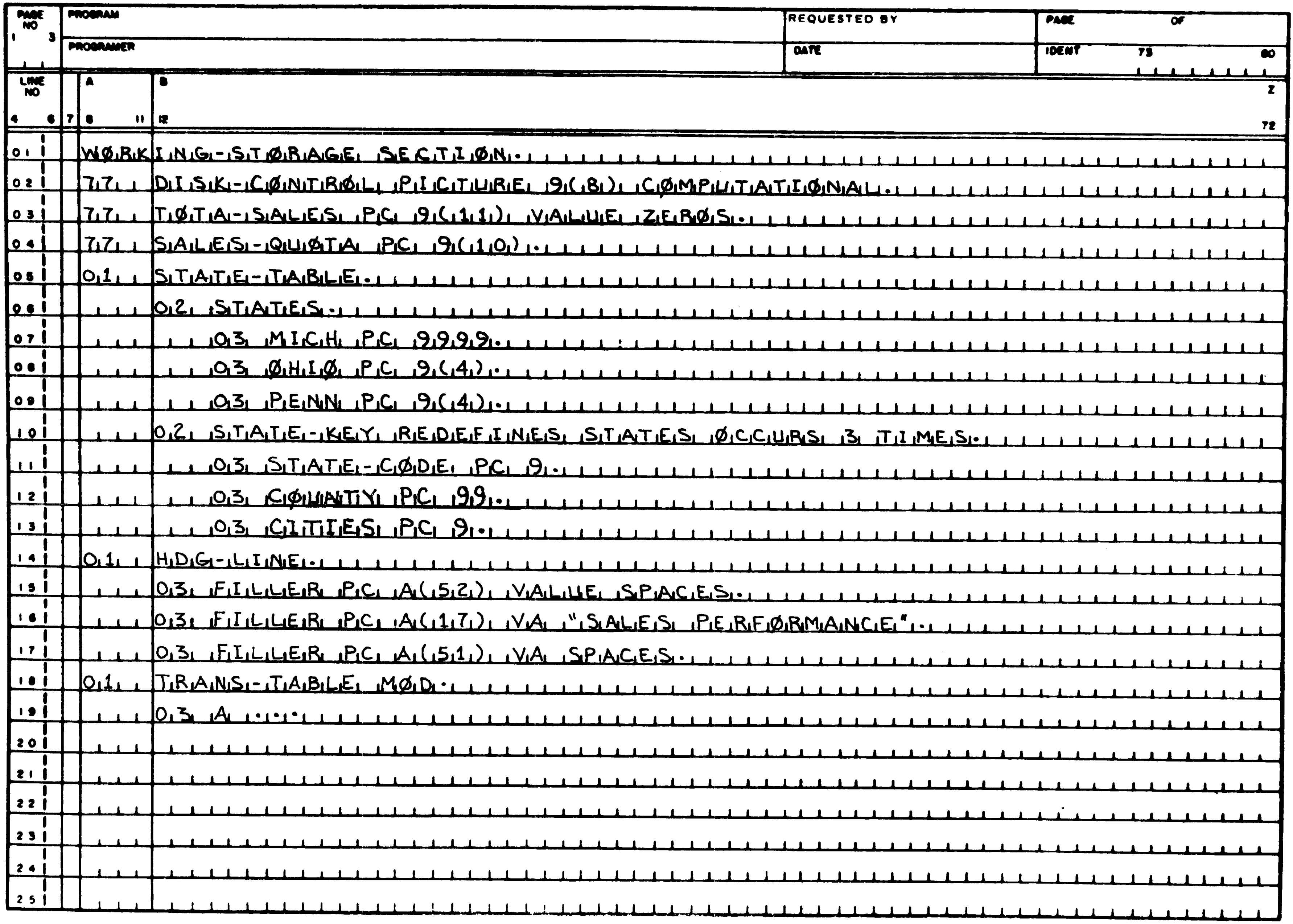
TRANSLATE TABLES.

If translate tables are desired in WORKING-STORAGE for referencing by the TRN operator in ENTER SYMBOLIC, the starting address must be modulo 1000. This can be accomplished by following the record-name with the reserved word MOD.

CODING THE WORKING-STORAGE SECTION.

Figure 4-6 illustrates the coding of the WORKING-STORAGE SECTION.

BURROUGHS COBOL CODING FORM



PAGE NO
PROGRAM
REQUESTED BY
PAGE OF
PROGRAMER
DATE
IDENT 73 80
LINE NO
A
B
Z
72

```
WØRKING-STØRAGE SECTIØN.
77  DISK-CØNTRØL PICTURE 9(8) CØMPUTATIØNAL.
77  TØTA-SALES PC 9(11) VALUE ZERØS.
77  SALES-QUØTA PC 9(10).
01  STATE-TABLE.
    02 STATES.
       03 MICH PC 9999.
       03 ØHIØ PC 9(4).
       03 PENN PC 9(4).
    02 STATE-KEY REDEFINES STATES ØCCURS 3 TIMES.
       03 STATE-CØDE PC 9.
       03 CØUNTY PC 99.
       03 CITIES PC 9.
01  HDG-LINE.
    03 FILLER PC A(52) VALUE SPACES.
    03 FILLER PC A(17) VA "SALES PERFØRMANCE".
    03 FILLER PC A(51) VA SPACES.
01  TRANS-TABLE MØD.
    03 A ....
```

Figure 4-6. Coding of WORKING-STORAGE SECTION

WORKING-STORAGE continued

SECTION 5

PROCEDURE DIVISION

GENERAL.

The fourth part of the COBOL source program is the PROCEDURE DIVISION. This division contains the procedures needed to solve a given problem. These procedures are written as sentences which may be combined to form paragraphs, which in turn may be combined to form sections.

RULES OF PROCEDURE FORMATION.

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, or a section, or a group of successive sections within the PROCEDURE DIVISION. The first entry following the PROCEDURE DIVISION header must be a section-name or a paragraph-name. If the first entry is a section-name, then it must be followed by a paragraph-name. Sentence structure is not governed by the rules of English grammar, but rather, dictated by the rules and formats outlined in this manual.

STATEMENTS.

There are three types of statements: imperative statements, conditional statements, and compiler-directing statements.

IMPERATIVE STATEMENTS

An imperative statement is any statement that is neither a conditional statement nor a compiler-directing statement. An imperative statement may consist of a sequence of imperative statements, each possibly separated from the next by a separator. A single imperative statement is made up of a verb followed by its operand. A sequence of imperative statements may contain either a GO TO statement or a STOP RUN statement which, if present, must appear as the last imperative statement of the sequence. Some of the imperative verbs are:

ACCEPT
ADD*
ALTER
CLOSE
COMPUTE*
DISPLAY
DIVIDE*
EXAMINE
EXIT
GO
INTERROGATE
MOVE
MULTIPLY*
OPEN
PERFORM
SEARCH
SEEK
SET
STOP
SUBTRACT*
WAIT
WRITE**

CONDITIONAL STATEMENTS.

A conditional statement specifies that a truth value of a condition is to be determined for subsequent action of the object program.

COMPILER-DIRECTING STATEMENTS.

A compiler-directing statement is one that consists of a compiler directing verb (COPY, ENTER and NOTE) and its operand(s).

SENTENCES.

There are three types of sentences: imperative sentences, conditional sentences, and compiler-directing sentences. A sentence consists of a sequence of one or more statements, the last of which is terminated by a period.

IMPERATIVE SENTENCES.

An imperative sentence is an imperative statement terminated by a period. An imperative sentence can contain either a GO TO statement or a STOP RUN statement which, if present, must be the last statement in the sentence. Examples would be:

```
     ADD MONTHLY-SALES TO TOTAL-SALES, THEN GO TO PRINT-TOTAL.
```

or

*   Without the SIZE ERROR option.

** Without the INVALID KEY option.

```
DISPLAY "PGM-END" THEN STOP RUN.
```

CONDITIONAL SENTENCES.

A conditional sentence is a conditional statement which may optionally contain an imperative statement and must always be terminated by a period.

EXAMPLES:

```
IF HEIGHT IS GREATER THAN SIX-FEET-NINE GO TO
TALL-MEN, ELSE ADD 1 TO PUNIES, GO GET-ANOTHER-
RECORD.

IF SALES IS EQUAL TO BOSSES-QUOTA THEN MOVE SALESMAN
TO HONOR-ROLL OTHERWISE MOVE HIS-NAME TO PINK-SLIP-
LIST, GO TO NEXT-SENTENCE.
```

If the phrase NEXT-SENTENCE immediately precedes a period, then the phrase may be eliminated and a GO TO NEXT-SENTENCE will be implied.

COMPILER-DIRECTING SENTENCES.

A compiler-directing sentence is a single compiler-directing statement terminated by a period.

EXAMPLE:

```
COPY "SCANER".
```

SENTENCE PUNCTUATION.

VERB FORMATS.

Punctuation rules for individual verbs are as shown in the verb formats and in section 1 of this manual.

SENTENCE FORMATS.

The following rules apply to the punctuation of sentences:

a. A sentence is terminated by a period.

b. A separator is a word or character used for the purpose of enhancing readability. The use of a separator (other than a space) is optional.

c. The allowable separators are: spaces, the semicolon (;), the comma (,), and the reserved word THEN.

d. Separators may be used in the following places:

   1) Between statements.
   2) In a conditional statement.

      a) Between the condition and statement-1.
      b) Between statement-1 and ELSE.

e. A separator (other than a space) should be followed by at least one space but is not required.

EXECUTION OF IMPERATIVE SENTENCES.

An imperative sentence is executed in its entirety and control is passed to the next applicable procedural sentence.

EXECUTION OF CONDITIONAL SENTENCES.

In the conditional sentence:

$$\underline{\text{IF}}\ \text{condition}\ \ \text{statement-1}\ \left\{\begin{matrix}\underline{\text{OTHERWISE}}\\ \underline{\text{ELSE}}\end{matrix}\right\}\ \text{statement-2}$$

the condition is an expression which is TRUE or FALSE. If the condition is TRUE, then statement-1 is executed and control is immediately transferred to the next sentence. If the condition is FALSE, statement-2 is executed and control passes to the next sentence.

If statement-1 is conditional, then the conditional statement must be the last (or only) statement comprising statement-1. For example, the conditional sentence would then have the form:

```
IF  condition-1  imperative-statement-1  IF  condition-2

statement-3   { OTHERWISE }   statement-4   { OTHERWISE }
              { ELSE      }                 { ELSE      }

statement-2.
```

If condition-1 is TRUE, imperative-statement-1 is executed. 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. Statement-3 can in turn be either imperative or conditional and, if conditional, can in turn contain conditional statements to an arbitrary depth. In an identical manner, statement-4 can either be imperative or conditional, as can statement-2. The execution of the phrase NEXT SENTENCE causes a transfer of control to the next sentence written in order, except when it appears in the last sentence of a procedure being PERFORMed, in which case control is passed to the return control.

EXECUTION OF COMPILER-DIRECTING SENTENCES.

The compiler-directing sentences direct activities during compilation time. On the other hand, procedural sentences denote action to be taken by the object program. Compiler-directing sentences may result in the inclusion of routines into the object program. They do not directly result in either the transfer or passing of control. The routines themselves, which the compiler-directing sentences may have included in the object program, are subject to the same rules for transfer or passing of control as if those routines had been created from procedural sentences only.

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 rules of the coding form (section 7), 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 may be varied simply by transferring to a named procedure.

In executing procedures, control is transferred only to the beginning of a paragraph or section. 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 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 may 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 coding form (Section 7), the source programmer begins a paragraph with a name. The name consists of a word followed by a period, and the name precedes the paragraph it names. A paragraph is terminated by the next paragraph-name. The smallest grouping of the PROCEDURE DIVISION which is named is a paragraph. The last paragraph in the PROCEDURE DIVISION is terminated by the optional special paragraph-name END-OF-JOB which can be the last card in the source program.

Programs may contain identical paragraph-names provided they are resident in different sections. If such paragraph-names are not qualified when used, the current section is assumed. They may be used in GO, PERFORM and ALTER statements if desired.

SECTIONS.

A section consists of one or more successive paragraphs and must be named when designated. The section-name is followed by the word SECTION, a priority number which is optional, and a period. If the section is a DECLARATIVE section, then the DECLARATIVE sentence (i.e., USE or COPY) follows the section header and begins on the same line. Under all other circumstances, a sentence may not begin on the same line as a section-name. 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 this technqiue is exceptionally useful in trimming down the physical size of object programs by stating a priority number to declare overlayable program storage (see SEGMENT CLASSIFICATION).

Since paragraph-names and section-names both have the same designated position on the reference format (i.e., position A), section-names, when specified, are written on one line followed by a paragraph name on a subsequent line. When PERFORM is used in a non-DECLARATIVE procedural section to call another section, the same rules apply as when PERFORM is used in a DECLARATIVE section.

DECLARATIVES.

Declaratives are procedures which operate under the control of the input-output system. Declaratives consist of compiler-directing sentences and their associated procedures. Declaratives, if used, must be grouped together, at the beginning of the PROCEDURE DIVISION. The group of declaratives must be preceded by the key word DECLARATIVES, and must be followed by the words END DECLARATIVES. Each DECLARATIVE consists of a single section and must conform to the rules for procedure formation. There are two statements that are called declarative statements in the COBOL Compiler. These are the USE and the COPY statements.

USE STATEMENT.

A USE declarative is used to supplement the standard procedures provided by the input-output system. The USE sentence, immediately following the section-name, identifies the condition calling for the execution of the USE procedures. Only the PERFORM statements may 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 as follows:

```
section-name SECTION.  USE...............
paragraph-name.  First procedure-statement ...
```

Complete rules for writing the formats for USE are stated under the USE verb.

COPY STATEMENT AS A DECLARATIVE.

A COPY declarative is used to incorporate a DECLARATIVE library routine in the source program. That is, a routine which is a USE declarative. The format of the COPY declarative is:

```
        section-name SECTION.  COPY "library-name".
```

Complete rules for writing the format for COPY are stated under the COPY verb.

ARITHMETIC EXPRESSIONS.

An arithmetic expression is an algebraic expression which is defined as:

a. An identifier of a numeric elementary item.

b. A numeric literal.

c. Such identifiers and literals separated by arithmetic operators.

d. Two arithmetic expressions separated by an arithmetic operator.

e. An arithmetic expression enclosed in parentheses.

Any arithmetic expression may be preceded by a unary + or -. The permissible combinations of identifiers, literals, and arithmetic operators are given in table 5-1. Those identifiers and literals appearing in an arithmetic expression must represent either numeric elementary items or numeric literals on which arithmetic operation may be performed.

Table 5-1

Combination of Symbols in Arithmetic Expressions

First Symbol	Second Symbol Variable	*/**	+-	(	)
Variable	-	P	P	-	P
*/**	P	-	P	P	-
+-	P	-	-	P	-
(	P	-	P	P	-
)	-	P	P	-	P

NOTE

In the above table, the letter P represents a permissable pair of symbols. The character - represents an invalid character pair. Variable represents an identifier or literal.

ARITHMETIC OPERATORS.

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

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

FORMATION AND EVALUATION RULES.

Parentheses may be used in arithmetic expressions to specify the order in which elements are to be used. Expressions within parentheses are evaluated first and, within a nest of parentheses, evaluation proceeds from the least inclusive set to the most inclusive set. When parentheses are not used, or parenthesized expressions are at the same level of inclusiveness, the following hierarchical order of operations is implied:

Unary + or -
**
* and /
+ and -

The symbols + and -, if used without parenthesizing, may only follow one of the arithmetic operators **, *, /, or appear as the first symbol in a formula. Parentheses have a precedence higher than any of the operators and are used to eliminate ambiguities in logic where consecutive operations of the same hierarchical level appear, or to modify the normal hierarchical sequence of execution in formulas where it is necessary to have some deviation from the normal precedence. When the sequence of execution is not specified by parentheses, the order of execution of consecutive operations of the same hierarchical level is from left to right. Thus, expressions ordinarily considered to be ambiguous, e.g., A / B * C, A / B / C, and A**B**C are permitted in COBOL. They are interpreted as if they were written (A / B) * C, (A / B) / C, and (A**B) **C, respectively. Without parenthesizing, the following example:

A + B / C + D ** E * F - G

would be interpreted as:

A + (B / C) + ((D ** E) * F) - G

with the sequence of operations working from the inner-most parentheses toward the outside, i.e., first exponentiation, then multiplication and division, and finally addition and subtraction.

The way in which operators, variables, and parentheses may be combined in an arithmetic expression is summarized in table 5-1.

An arithmetic expression may only begin with the symbols (, +, -, or a variable and may only end with a) or a variable. There must be a one-to-one correspondence between left and right parenthesis of an arithmetic expression such that each left parenthesis is to the left of its corresponding right parenthesis.

CONDITIONS.

A condition causes the object program to select between alternate paths of control depending upon the truth value of a test. Conditions are used in IF and PERFORM statements. A condition is one of the following:

a. Relation condition.
b. Class condition.
c. Condition-name condition.
d. Sign condition.
e. NOT condition.
f. Condition $\left\{ \begin{matrix} \underline{\text{AND}} \\ \underline{\text{OR}} \end{matrix} \right\}$ condition.

The construction NOT condition, where condition is one of the first four types of conditions listed above, is not permitted if the condition itself contains NOT.

LOGICAL OPERATORS.

Conditions may be combined by logical operators. The logical operators must be preceded by a space and followed by a space. The meaning of the logical operators is as follows:

Logical Operator	Meaning
OR	Logical Inclusive OR
AND	Logical Conjunction
NOT	Logical Negation

Table 5-2 indicates the relationships between the logical operators and conditions A and B. Table 5-3 indicates the way in which conditions and logical operators may be combined.

RELATION CONDITION.

A relation condition causes comparison of two operands, each of which may be a data-name, a literal, or an arithmetic expression (formula). Comparison of two elementary numeric items is permitted regardless of the format as specified in individual USAGE clauses. However, for all other comparisons the operands must have the same USAGE. Group numeric items are defined to be alphanumeric. It is not permissible to compare an index-data-name to a literal or a data-name.

Table 5-2

Relationship of Conditions, Logical Operators, and Truth Values

Condition		Condition and Value		
A	B	A AND B	A OR B	NOT A
TRUE	TRUE	TRUE	TRUE	FALSE
FALSE	TRUE	FALSE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE	FALSE
FALSE	FALSE	FALSE	FALSE	TRUE

Table 5-3

Combinations of Conditions and Logical Operators

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

NOTE

The letter P represents a permitted pair of symbols. The character - represents an invalid character pair.

The general format for a relation condition is as follows:

$$\left\{\begin{array}{l}\text{data-name-1}\\ \text{literal-1}\\ \text{arith. expression-1}\end{array}\right\}\ \text{relational-operator}\ \left\{\begin{array}{l}\text{data-name-2}\\ \text{literal-2}\\ \text{arith. expression-2}\end{array}\right\}$$

The first operand, data-name-1, literal-1, or arithmetic expression-1 is called the subject of the condition. The second operand, data-name-2, literal-2, arithmetic expression-2 is called the object of the condition. The object and the subject may not both be literals.

RELATIONAL OPERATORS.
The relational operators specify the type of comparison to be made in a relation condition. The relational operators must be preceded by a space and followed by a space. Relational operators are:

a. IS [NOT] GREATER THAN.
b. IS [NOT] LESS THAN.
c. IS [NOT] EQUAL TO.
d. IS [NOT] >.
e. IS [NOT] <.
f. IS [NOT] =.

COMPARISON OF OPERANDS.
NON-NUMERIC. For non-numeric (byte) operands, a comparison will result when determination is made that one operand is less than, equal to, or greater than the other with respect to a specified internal coding sequence of characters (see appendix C). The size of an operand is the total number of characters or digits in the operand. Non-numeric operands may be compared only when their USAGE is the same, implicitly or explicitly. There are two cases to consider:

a. If the operands are of equal size, characters or digits in corresponding character or digit positions of the two operands are compared starting from the high-order end through the low-order end. If all pairs of characters or digits compare equally through the last pair, the operands are considered equal when the low-order end is reached. The first pair of unequal characters or digits to be encountered is compared to determine their respective relationship. The operand that contains the character or digit that is positioned higher in the internal coding sequence is considered to be the greater operand.

b. If the operands are of unequal size, the comparison of characters or digits proceeds from high-order to low-order positions until a pair of unequal characters or digits is encountered, or until one of the operands has no more characters or digits to compare. If the end of the shorter operand is reached and the remaining characters or digits in the longer operand are spaces or zeros, the two operands are considered to be equal.

NUMERIC. For operands that are numeric, a comparison results in the determination that one of them is less than, equal to, or greater than the other with respect to the algebraic value of the operands. The length of the operands, in terms of number of digits, is not significant. Zero is considered a unique value regardless of the sign. Comparison of these operands is permitted regardless of the manner in which their usage is described. Unsigned numeric operands are considered positive for purposes of comparisons.

The signs of signed numeric operands will be compared as to their algebraic value of being plus (highest) or minus (lowest).

EVALUATION RULES.

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

a. Arithmetic expressions (formulas).

b. All relational operators.

c. NOT.

d. AND.

e. OR.

SIMPLE CONDITIONS.

Simple conditions, as distinguished from compound conditions, are subdivided into four general families of conditional tests: Relation Tests, Relative Value Tests, Class Tests, and the Conditional Variable Tests. A detailed explanation of each of these can be found under the IF verb discussion.

COMPOUND CONDITIONS.

The most common format of a compound condition is:

$$\text{simple-condition-1} \quad \left\{ \begin{matrix} \underline{\text{AND}} \\ \underline{\text{OR}} \end{matrix} \right\} \quad \text{simple-condition-2}$$

$$\left[\left\{ \begin{matrix} \underline{\text{AND}} \\ \underline{\text{OR}} \end{matrix} \right\} \quad \cdots \quad \left\{ \begin{matrix} \underline{\text{AND}} \\ \underline{\text{OR}} \end{matrix} \right\} \quad \text{simple-condition-n} \right]$$

Simple conditions can be combined with logical operators according to specified rules to form compound conditions. The logical operators AND, OR, and NOT are shown in table 5-2 where A and B represent simple conditions. Thus, if A is TRUE and B is FALSE, then the expression A AND B is FALSE, while the expression A OR B is TRUE.

The following are illustrations of compound conditions:

a. AGE IS LESS THAN MAX-AGE AND AGE IS GREATER THAN 20.

b. AGE IS GREATER THAN 24 OR MARRIED.

c. STOCK-ON-HAND IS LESS THAN DEMAND OR STK-SUPPLY IS GREATER THAN DEMAND + INVENTORY.

d. A IS EQUAL TO B, AND C IS NOT EQUAL TO D, OR E IS NOT EQUAL TO F, AND G IS POSITIVE, OR H IS LESS THAN I * J.

e. STK-ACCT IS GREATER THAN 72 AND (STK-NUMBER IS LESS THAN 100 OR STK-NUMBER EQUAL TO 76920).

Note that it is not necessary to use the same logical connective throughout. The rules for determining the logical (i.e., truth) value of a compound condition are as follows:

a. If AND's are the only logical connectives used, then the compound condition is TRUE if, and only if, each of the simple conditions is TRUE.

b. If OR's are the only logical connectives used, then the compound condition is TRUE if, and only if, one or more of the simple conditions is TRUE.

c. If both logical connectives are used, then the conditions are grouped first according to AND, proceeding from left to right, and then by OR, proceeding from left to right.

Parentheses may be used to indicate grouping as specified in the examples below. Parentheses must always be paired the same as in algebra, i.e., the expressions within the parentheses will be evaluated first. In the event that nested parenthetical expressions are employed, the innermost expressions within parentheses are handled first. Examples of using parentheses to indicate grouping are:

a. To evaluate C1 and (C2 OR NOT (C3 OR C4)), use the first part of rule c above and successively reduce this by substituting as follows:

Let C5 equal "C3 OR C4" resulting in
C1 AND (C2 OR NOT C5)

Let C6 equal "C2 OR NOT C5" resulting
in C1 AND C6

This can be evaluated by table 5-2.

b. To evaluate C1 OR C2 AND C3, use the second part of rule c and reduce this to C1 OR (C2 AND C3), which can now be reduced as in example a.

c. To evaluate C1 AND C2 OR NOT C3 AND C4, group first by AND from left to right, resulting in:

(C1 AND C2) OR (NOT C3 AND C4)

which can now be evaluated as in example a.

d. To evaluate C1 AND C2 AND C3 OR C4 OR C5 AND C6 AND C7 OR C8, group from the left by AND to produce:

((C1 AND C2) AND C3) OR C4 OR ((C5 AND C6)
AND C7) OR C8

which can now be evaluated as in example a.

ABBREVIATED COMPOUND CONDITIONS.

Any relation condition other than the first that appears in a compound conditional statement may be abbreviated as follows:

a. The subject, or the subject and relational operator, may be omitted. In these cases, the effect of the abbreviated relation condition is the same as if the omitted parts had been taken from the nearest preceding complete relation condition within the same condition. That is, the first relation in a condition must be complete.

b. If, in a consecutive sequence of relation conditions (separated by logical operators) the subjects are identical, the relational operators are identical and the logical connectors are identical, the sequence may be abbreviated as follows:

1) Abbreviation 1 - when identical subjects are omitted in a consecutive sequence of relation conditions. An example of Abbreviation 1 would be:

IF A = B AND = C.

This is equivalent to IF A = B AND A = C.

2) Abbreviation 2 - when identical subjects and relational operators are omitted in a consecutive sequence of relation conditions. An example of Abbreviation 2 is:

IF A = B AND C.

This is equivalent to IF A = B AND A = C.

3) Abbreviation 3 - when identical subjects, relational operators, and logical connectors are omitted in a consecutive sequence of relational conditions. Only the first occurrence of the subject and relation are written; all objects but the last are written as a series (can but need not be preceded by commas). The logical connector is written only once and appears immediately preceding the last of the objects. An example of Abbreviation 3 would be:

IF A = B, C AND D

This is equivalent to IF A = B AND A = C AND A = D.

c. As indicated in the previous paragraphs, compound conditions can be abbreviated by having implied subjects, or implied subjects and relational operators, providing the first simple condition is a full relation. The missing term is obtained from the last previous complete relation in the sentence. The following examples further illustrate the abbreviated compound conditions:

1) IF A = B OR C is equivalent to IF A = B OR A = C.

2) IF A < B OR = C OR D is equivalent to IF A < B OR A = C OR A < D. Note that the missing relational symbol for D is < rather than = since the last complete relation is A < B.

SEGMENTATION.

COBOL segmentation is a facility that provides a means by which communication with the compiler, to specify object program overlay requirements, can be accomplished. COBOL segmentation deals only with segmentation of procedures. As such, only the PROCEDURE DIVISION and the ENVIRONMENT DIVISION are considered in determining segmentation requirements for an object program.

PROGRAM SEGMENTS.

Although it is not mandatory, the PROCEDURE DIVISION for a source program may be written as a consecutive group of sections, each of which are operations that are designed to collectively perform a particular function. Each section must be classified as belonging either to the fixed portion or to one of the independent segments of the object program. Segmentation in no way affects the need for qualification of procedure-names to ensure uniqueness.

The object program is composed of two types of segments: A fixed segment and overlayable segments.

a. The fixed segment is the main program segment and is never overlaid by any other part of the program.

b. An overlayable segment is a segment which, although logically treated as if it were always in memory, can be overlaid, if necessary, by another segment to optimize memory utilization. However, such a segment, if called for by the program, is always made available in its "initial" state except for ALTERed switches which are always set to their last used state.

Also, depending on availability of memory, the number of permanent segments in the fixed and overlayable portions can be varied by changing the SEGMENT-LIMIT clause in the OBJECT-COMPUTER paragraph.

SEGMENT CLASSIFICATION.

Sections which are to be segmented are classified using a system of priority numbers and the following criteria:

a. Logic requirements - sections with priority numbers from 00 thru 49 in a program may reside in the fixed segment depending on the value specified in SEGMENT-LIMIT. Sections containing a priority number lower than that specified in SEGMENT-LIMIT, regardless of their physical location in the program, will be assigned to the fixed segment; all other sections will be assigned as overlayable segments. Fall-through control from one SECTION to another SECTION is accomplished in their order of appearance in the source program.

b. Relationship to other sections - sections coded within the SEGMENT-LIMIT range will become the fixed segment and can communicate freely with each other. Those coded outside the stated SEGMENT-LIMIT range fall into the overlayable category and can also communicate from one to the other, except that a PERFORM may not have within its range any procedure-name contained in another overlayable segment. The compiler will create one non-overlayable (fixed) program area which will include all sections with priority numbers below the value specified in SEGMENT-LIMIT. One overlayable area in memory, the size of the largest declared section, will be created for priority numbers equal to, or higher than, the value specified in SEGMENT-LIMIT. This method allows the smaller overlays to be called in without requiring memory alignment.

PRIORITY NUMBERS.

Section overlay classifications are accomplished by means of a system of priority numbers. The priority number is included in the section header. The general format of a section header is as follows:

```
section-name   SECTION   priority-number.
```

The priority number must be an integer ranging in value from 00 through 99 (also 0, 1, 2, etc., are permissible priority numbers).

If the priority number is omitted from the section header, the priority number is assumed to be 0. Segments with priority numbers ranging from 0 up to, but not including, the value specified in the SEGMENT-LIMIT clause (or 49 if no SEGMENT-LIMIT clause has been specified) are considered as being located in the fixed (non-overlayable) portion of the object program. Segments with priority numbers equal to or higher than, the value specified in SEGMENT-LIMIT, but not exceeding 99, are independent segments (overlayable) and fully ALTERable. Sections in DECLARATIVES are assumed to be 00 and must not contain priority numbers in their section headers. Priority numbers may be stated in any sequence and need not be in direct sequence. The fixed segment does not end when the first priority number equal to or greater than SEGMENT-LIMIT is encountered.

All segments, regardless of their physical location in the source program, whose priority number is less than that which is specified in SEGMENT-LIMIT will be "gathered" into a single non-overlayable segment. All other segments equal to, or greater than that which is specified in SEGMENT LIMIT will be "gathered" into overlayable segments according to equal priority numbers regardless of their physical location in the source program.

The use of the "gathering" technique will allow programmers to create tailored segments which will reduce disk access times. For example:

Program A: SEGMENT-LIMIT equals 17.

Non-Gathered

Segment	Description	Size in Digits
00-16	Main body of the program	20,000
17	Used frequently	1,000
18	Used frequently	5,000
19	Used infrequently	4,000
20	Used at EOJ only	500
21	Used frequently	2,000

Segment	Description	Size in Digits
22	Used at BOJ only	1,000
23	Used frequently	500
24	Used if TRACE desired	1,500
25	Used infrequently	3,000

Gathered

Segment	Description	Size in Digits
00-16	Main body of the program	20,000
17	Used frequently	1,000
18	Used infrequently	5,000
19	Used infrequently	4,000
20	Used at EOJ	500
17	Used frequently (was segment 21)	2,000
19	Used at BOJ (was segment 22)	1,000
17	Used frequently (was segment 23)	500
20	Used if TRACE desired (was segment 24)	1,500
20	Used infrequently (was segment 25)	3,000

Results of Gathering

Segment	Description	Size in Digits
00-16	Main body of the program	20,000
17	Used frequently	3,500
18	Used infrequently	5,000
19	Used infrequently	5,000
20	Used infrequently	5,000

"Fall through" will be performed in the sequence as outlined in the above Non-Gathered example and not as they appear in the Results of Gathering example above, therefore preserving the logical integrity of the original program.

The Burroughs unique head-per-track disk file permits B 2500/B 3500 Systems users to efficiently handle the COBOL technique of overlaying segments without requiring a programmer to state varied hardware inadequacies and is the decided factor by which the B 2500/ B 3500 excels in its multiprocessing capabilities.

The MCP will automatically check to see if an overlay being called for by an object program is already present in the object programs overlayable memory storage area. If it is present, no disk access is required and the program is not interrupted. If it is not present, the MCP interrupts the program and will access the disk for the desired overlayable portion of the program. The MCP uses overlay segments directly from the program library where the object program was compiled to and is called in as an overlay in its <u>initial generated code each and every time it is required</u> by the operating program. Although the initial code is retrieved each time, the latest addresses of ALTERed exits are still applicable and are in force by the use of an automatic ALTER table.

INTERNAL PROGRAM SWITCHES.

Every compiled object program contains eight programmatic switches provided automatically. Switches SW1 through SW7 are composed of one unsigned digit in length and are located in memory locations (base relative) 1 through 7. SW8 is located in memory location (base relative) 0. These switches can be set optionally as follows:

a. Option 1. Switches can be initially set at the start of an object program's execution by punching one of the following MCP Control Cards:

    1) ? EXECUTE program-name VALUE 0 = nnnnnn*
    2) ? EXECUTE program-name VALUE 1 = nnnnnn**
    3) ? EXECUTE program-name VALUE 2 = nnnnnn***

    nnnnnn represents internal program switches which may be set by placing a zero (OFF) or any digit 1-9 (ON) in the appropriate positions.

* VALUE 0 sets switches 812345 as coded in nnnnnn.
** VALUE 1 sets switches 123456 as coded in nnnnnn.
***VALUE 2 sets switches 234567 as coded in nnnnnn.

Note that the VALUE statement in an MCP control message must always contain a six digit integer, thus only six switches can be affected using this method. The program-name must be the object program identifier.

b. Option 2. Switches can be referred to in the PROCEDURE DIVISION by the use of the reserved words SW1, SW2... SW8. Each individual switch setting can be changed during operation by a MOVE, ADD, SUBTRACT, etc., for example:

```
MOVE 0 TO SW1.
ADD 1 TO SW2.
SUBTRACT 1 FROM SW3.
```

c. Option 3. The systems operator can be programmatically informed that a switch requires setting by stating in the source program:

```
STOP "SET SW2 ON".
```

The proper keyboard entry to set the switch would be:

```
mix-index IN 2 1 UN = 1
```

The proper keyboard entry to resume operation would be:

```
mix-index AXGO
```

Note that SW6 has an affect on the MONITOR DEPENDING....requirement if the statement is present.

The switch memory locations are reserved and operate exactly like the reserved TALLY locations.

All three options of setting switches can be incorporated during the operation of a given program.

VERBS.

Some of the verbs available for use with the COBOL Compiler are categorized below. Although the word IF is not a verb in the

English language, it is utilized as such in the COBOL language. Its occurrence is a vital feature in the PROCEDURE DIVISION.

a. Arithmetic:

ADD
SUBTRACT
MULTIPLY
DIVIDE
COMPUTE

b. Compiler directing declaratives:

NOTE
USE

c. Compiler directing:

COPY

d. Data manipulations:

MOVE
EXAMINE
SORT

e. Ending:

STOP

f. Input-output:

WRITE
READ
OPEN
CLOSE
ACCEPT
DISPLAY
SEEK

g. Logical Control:

IF

h. Procedure Branching:

GO

ALTER
PERFORM
EXIT
ZIP

i. Source-level Debugging:
TRACE

SPECIFIC VERB FORMATS.

The specific verb formats, together with a detailed discussion of the restrictions and limitations associated with each, appear on the following pages in alphabetic sequence.

ACCEPT.

The function of this verb is to permit the entry of low-volume data from the console typewriter.

The construct of this verb is:

```
                                 ┌ SPO           ┐
ACCEPT  data-name  [  FROM      <                 >  ]
                                 └ mnemonic-name ┘
```

This statement causes the operating object program to halt and wait for appropriate data to be entered on the SUPERVISORY PRINTER (SPO). The SPO entry will replace the contents of memory specified by the data-name. The systems operator answers an ACCEPT halt by keying in the following message:

mix-index AXdata-required

If a blank appears between the AX and data-required, the blank character will be included in the data-stream.

The number of characters ACCEPTed must correspond to the size of the receiving data-name.

If mnemonic-name is used, it must appear in the SPECIAL-NAMES paragraph equated to the hardware-name SPO.

The receiving data-name may be a group level entry and cannot be subscripted.

Because of the inefficiency of entering data through the keyboard, this technique of data transmission should be solely restricted to low-volume input data.

The maximum number of characters per ACCEPT statement is unlimited.

ACCEPT's of greater than 60 characters must be entered thru the SPO in exact groups of 60 characters, except for the last group, which can be of any size up to 60.

ADD.

The function of this verb is to add two or more numeric data items and adjust the value of the receiving field(s) accordingly.

The construct of this verb has four options:

Option 1:

```
ADD   { literal-1   }  [ { literal-2   }      ... ]
      { data-name-1 }  [ { data-name-2 }          ]

data-name-n [ROUNDED]  [ ON SIZE ERROR  any statement
                       [
[ { OTHERWISE }    statement ] ]
[ { ELSE      }              ] ]
```

Option 2:

```
ADD   { literal-1   }  [ { literal-2   }      ... ]
      { data-name-1 }  [ { data-name-2 }          ]

TO  data-name-3 [ROUNDED]   [ data-name-n [ROUNDED] ... ]

[ ON SIZE ERROR any statement   [ { OTHERWISE }    statement ] ]
[                               [ { ELSE      }              ] ]
```

Option 3:

```
ADD   { literal-1   }   { literal-2   }   [ { literal-3   }  ... ]
      { data-name-1 }   { data-name-2 }   [ { data-name-3 }      ]

GIVING  data-name-n  [ROUNDED]    [ ON SIZE ERROR any
                                  [

statement    [ { OTHERWISE }    statement ] ]
             [ { ELSE      }              ] ]
```

Option 4:

```
ADD  { CORR          }  data-name-1 TO data-name-2 [ROUNDED]
     { CORRESPONDING }

[ON SIZE ERROR any statement  [{ OTHERWISE }   statement ] ]
                               { ELSE      }
```

If Option 1 is used, the operands will be added together and the sum will be stored as the value of the last operand.

With Option 2, the value(s) of the operand(s) preceding the word TO will be added together and the sum will be added to the existing value(s) of operand(s) following the word TO. A resumation does not occur if the value of one of the data-names changes in the process. For example:

```
ADD A TO B,A,C.
```

In Option 3, the sum of the operands preceding the word GIVING will be inserted as a replacement value of data-name following the word GIVING.

In Options 1, 2, and 3, the data-names must refer to elementary numeric items only, except that data-names appearing only to the right of the word GIVING may refer to data-names which contain editing symbols.

An ADD statement must have at least two operands.

Editing items can only be used as the receiving field with the GIVING format. Operational signs and implied decimal points are not considered as editing symbols.

The composite of operands, which is that data item resulting from the superimposition of all operands, excluding the data item that follows the word GIVING, aligned on their decimal points, must not contain more than 98 digits/characters.

The internal format of operands referred to in an ADD statement may differ among each other. Any necessary format transformation and decimal point alignment is automatically supplied throughout the calculation.

Each literal must be a numeric literal.

If, after point alignment with the receiving data item, the calculated result would extend to the right of the receiving data item (i.e., a data-name whose value is to be set equal to the sum), truncation will occur. Truncation is always in accordance with the size associated with the resultant data-name. When the ROUNDED option is specified, it causes the resultant data-name to have its absolute value increased by 1 whenever the most-significant digit of the truncated portion is greater than or equal to five.

Whenever the magnitude of the calculated result exceeds the largest magnitude that can be contained in a resultant data-name, a size error condition arises. In the event of a size error condition, one of two possibilities will occur, depending on whether or not the ON SIZE ERROR option has been specified. The testing for the size error condition occurs only when the ON SIZE ERROR option has been specified.

a. In the event that ON SIZE ERROR is not specified and size error conditions arise, the value of the resultant data-name is unpredictable.

b. If the ON SIZE ERROR option has been specified and size error conditions arise, then the value of the resultant data-name will not be altered. After determining that there is a size error condition, the "any imperative-statement" associated with the ON SIZE ERROR option will be executed.

If Option 4 is used, multiple operations are performed. The operations are executed by pairing identical 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. All general rules pertaining to the ADD verb apply to each individual ADD operation. For instance, if the size of matched data-names does not correspond in that the decimal point is out of alignment or the sizes differ, the decimal point alignment or truncation takes place according to the rules previously discussed.

In the process of pairing identical data-names, any data-name with the REDEFINES clause is ignored. Similarly, data-names which are subordinate to the subordinate data-names with the REDEFINES clause are ignored.

NOTE

> This restriction does not preclude data-name-1 or data-name-2 themselves from having REDEFINES clauses or from being subordinate to data-names with REDEFINES clauses.

If the CORR or CORRESPONDING option is used, no item in the group referred to can contain an OCCURS clause.

If, in Option 4, either data-name-1 or data-name-2 is a group item which contains RENAMES entries, the entries are not considered in the matching of names.

In Option 4, data-name-1 and data-name-2 must not have a level number of 66, 77, or 88.

In Option 4, CORR is an acceptable substitute for CORRESPONDING.

ALTER.

The function of this verb is to modify a predetermined sequence of operations by changing the operand of a labeled GO TO paragraph.

The construct of this verb is:

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

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

Procedure-name-1, procedure-name-3, ... are names of paragraphs, each of which contains a single sentence consisting of only a GO TO statement as defined under Option 1 of the GO TO verb. Procedure-name-2, procedure-name-4, ... are not subject to the same restrictions and thus may be either paragraph names or section names.

When control passes to procedure-name-1, control is immediately passed to procedure-name-2 rather than to the procedure-name referred to by the GO TO statement in procedure-name-1. Procedure-name-1 is therefore a "gate" which remains set until again referenced by another ALTER statement.

Segmentation does not affect ALTER. Any GO TO paragraph may be ALTERed from anywhere in the program to PROCEED to any section or paragraph-name contained in the program.

CLOSE.

The function of this verb is to communicate to the MCP that the designated file-name being operated on or created is programmatically completed, and also to fulfill the stated action requirements.

The construct of this verb is:

```
                                  ┌            ┌ LOCK       ┐ ┐
CLOSE  file-name-1 [REEL]         │  WITH      │ PURGE      │ │
                                  │            │ RELEASE    │ │
                                  └            └ NO REWIND  ┘ ┘

[file-name-2...]
```

File-names must not be those defined as being SORT files.

The file must have been opened previously before a CLOSE statement can be executed.

This statement applies to the following categories of input and output files:

a. Files whose input and output media involve print files, card files, etc.

b. Files which are contained entirely on one reel and are the only files on that reel.

c. Files which may be contained on more than one physical reel. Furthermore, the number of reels might possibly be higher than the number of physical tape units provided on the system.

d. Disk files.

To show the effects of the CLOSE options, each type of file will be discussed separately.

a. Card and MICR Input.

1) CLOSE - releases the input areas, but does not release the reader.

2) CLOSE WITH NO REWIND - same as CLOSE.

3) CLOSE WITH RELEASE - releases the input areas and returns the reader to the MCP.

4) CLOSE WITH LOCK - same as CLOSE WITH RELEASE.

5) CLOSE WITH PURGE - same as CLOSE WITH RELEASE.

b. Card Output.

1) CLOSE - punches the trailer label (if any), releases the output areas, but does not release the punch.

2) CLOSE WITH NO REWIND - same as CLOSE.

3) CLOSE WITH RELEASE - releases the output areas and returns the punch to the MCP.

4) CLOSE WITH LOCK - same as CLOSE WITH RELEASE.

5) CLOSE WITH PURGE - same as CLOSE WITH RELEASE.

c. Tape Input.

1) CLOSE - checks the trailer label (if any) and rewinds the tape. It does not release input areas, and the unit remains assigned to the program.

2) CLOSE WITH NO REWIND - same as CLOSE except the tape is not rewound.

3) CLOSE WITH LOCK - releases the input areas, checks the trailer label (if any), rewinds the tape, and the MCP marks the unit not ready.

4) CLOSE WITH RELEASE - releases the input areas. checks the trailer label (if any), rewinds the tape, and returns the unit to the MCP.

5) CLOSE WITH PURGE - releases the input areas, checks the trailer label (if any), rewinds the tape, and if a write ring is in the reel, over-writes the label, making the tape a scratch tape.

d. Tape Output.

1) CLOSE - writes the trailer label (if any), and rewinds the tape. The unit remains assigned to the program.

2) CLOSE WITH NO REWIND - writes the trailer label (if any). The tape remains positioned beyond the trailer label (or tape mark if there is no trailer label). The unit remains assigned to the program.

3) CLOSE WITH LOCK - releases the output areas, writes the trailer label (if any), rewinds the tape, and the MCP marks the unit not ready.

4) CLOSE WITH RELEASE - releases the output areas, writes the trailer label (if any), rewinds the tape, and returns the unit to the MCP.

5) CLOSE WITH PURGE - releases the output areas, writes the trailer label (if any), rewinds the tape, returns the unit to the MCP, and the MCP over-writes the label making it a scratch tape.

e. Printer and Lister Output.

1) CLOSE - prints the trailer label (if any), releases the output areas but does not release the printer or lister.

2) CLOSE WITH NO REWIND - same as CLOSE.

3) CLOSE WITH RELEASE - releases the output areas and returns the printer or lister to the MCP.

4) CLOSE WITH LOCK - same as CLOSE WITH RELEASE.

5) CLOSE WITH PURGE - same as CLOSE WITH RELEASE.

f. Disk Files. The actions taken on files ASSIGNED to DISK will be discussed in terms of old files and new files. An old file is one that already exists on disk and appears in the MCP Disk Directory. A new file is one created by the program and does not appear in the Directory. A new file may only be referenced by the program which creates it.

1) CLOSE.

a) For an old file, the file is left in the Directory and is available to other programs.

b) For a new file, the file is not entered in the Directory, however, it remains on the disk and may be OPENed again by this program.

2) CLOSE WITH NO REWIND - not permitted on disk files.

3) CLOSE WITH RELEASE.

a) For an old file, same as CLOSE file-name.

b) For a new file, the file is entered in the Directory (thereby making it an old file). The file is available to be OPENed by any program.

4) CLOSE WITH LOCK.

a) For an old file, the file remains in the Directory and is made available.

b) A new file is entered in the Directory. Subsequent action is identical to an old file.

5) CLOSE WITH PURGE.

a) An old file is immediately removed from the disk and deleted from the Directory.

b) A new file will be immediately removed from the disk.

g. Remote Devices (Data Communications).

1) CLOSE - releases the input areas, but does not release the adapter.

2) CLOSE WITH RELEASE - releases the input areas and returns the remote device to the system.

If a file has been specified as being OPTIONAL, the standard END-OF-FILE processing is not permitted whenever the file is not present.

If a CLOSE statement without the REEL option has been executed for a file, a READ, WRITE, or SEEK statement for that file must not be executed unless an intervening OPEN statement for that file is executed.

The CLOSE REEL option signifies that the file-name being CLOSEd is a multi-reel magnetic tape input/output file. The reel will be CLOSEd at the time of encountering the CLOSE REEL statement and an automatic OPEN of the next sequential reel of the multi-reel file will be performed by the MCP.

COMPUTE.

The function of this verb is to assign to a data item the value of a numeric data item, literal, or arithmetic expression.

The construct of this verb is:

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

[ON SIZE ERROR any statement  [ { OTHERWISE }      statement ] ]
                                { ELSE      }
```

The literal must be numeric literal.

Data-name-2 must refer to an elementary numeric item. Data-name-1 may describe a data item which contains editing symbols.

The arithmetic expression option permits the use of any meaningful combination of data-names, numeric literals, arithmetic operators, and parenthesization, as required.

The maximum size of an operand is 99 decimal digits.

All rules regarding ON SIZE ERROR, ROUNDED options, truncation and editing are the same as for ADD.

If numeric-literal exponents are used, the results are accurate up to 18 digits in length or to as many decimal places.

If data-name-2 exponent is used, the accuracy of the result is dependent upon whether or not; and the manner in which, the result is rounded, truncated and/or defined. An example of a numeric literal exponent would be,

```
COMPUTE X = A**2
```

where A is equal to 4. The result will be the X equals 16.

An example of data-name-2 exponent would be,

```
COMPUTE X = A**B
```

where A equals 4 and B equals 2. The result will be that X equals 15.999 if X is defined as PC 99.999 or X equals 15 if X is defined as PC 99.

An example of data-name-2 exponent (ROUNDED) would be,

```
COMPUTE X ROUNDED = A**B
```

where A equals 4 and B equals 2. The result will be that X = 16 if X is defined as PC 99 or X equals 16.000 if X is defined as PC 99.999.

When data-name-1 is specified as being ROUNDED, regardless of the decimal point location, the result will reflect greater accuracy.

NOTE

The 17 KB version of the COBOL Compiler will accept only integer numeric-literal exponents consisting of six digits or less in length.

The 30 KB version of the COBOL Compiler will accept the entire COMPUTE construct, however, the use of exponents comprised of data-name-2, fractionalized literals or numeric-literals longer than six digits require the presence of floating point hardware.

COPY.

The function of this verb is to allow library routines contained on a source language library file to be incorporated into the program.

The construct of this verb contains two options which are:

Option 1:

```
COPY   library-name .
```

Option 2:

```
COPY   library-name

[ REPLACING     { word-1      }   BY   { word-2      }
                { data-name-1 }        { data-name-2 }

              [ { word-3      }   BY   { word-4      }  ... ] . ]
                { data-name-3 }        { data-name-4 }
```

The COPY statement may refer only to one library entry in the library for every time it is used. Library-name is the value placed in a library entry bounded by quotes or a procedure-name type word. The library entry bounded by quotes cannot contain more than six characters, where the procedure-name entry may be more than six characters, however, if the procedure-name type entry is greater than six characters, only the most significant six will be used for the library-name. If the library-name is a procedure-name type word and is numeric it must be separated from the period (if present) by a space.

The library file is inserted in the source program immediately after the COPY statement at compilation time. The result is the same as if the library data were actually a part of the source program.

Library data can encompass an entire procedure which may be any number of statements, paragraphs, or entire source program divisions or parts thereof.

Library files may not contain COPY statements.

No statement may appear to the right of the COPY statement on the same source card.

COPY during the PROCEDURE or ENVIRONMENT divisions must follow a SECTION or paragraph-name and all information contained in the library file is included and can be fully referenced.

On a COPY during the DATA DIVISION, the FD file-name, or the level 01 data-name preceding the COPY is saved and the relative constructs from the library file are discarded. For example, the statement

```
FD MASTER-INPUT COPY "MASTER".
```

will cause the library file titled MASTER to be inserted into the source program immediately following the COPY statement. The source program must refer to the FD file-name as MASTER-INPUT and not as MASTER. The library FD file-name will appear on the output listing, but cannot be referenced in the source program.

Library files copied from the library are flagged on the output listing by an L preceding the sequence number.

In Option 2, a word is defined as being any COBOL word that is not a COBOL Reserved Word. For example, the following statement reflects non-reserved COBOL words AAA,BBB and 1234, where AAA and BBB are data-names and 1234 is a COBOL word:

```
MULTIPLY AAA BY BBB, THEN GO TO 1234.
```

If the COPY REPLACING option is specified, each word-1 or data-name-1 stipulated will be replaced BY the word-2 or data-name-2 entries specified in the option. Data-names may not be subscripted, indexed or qualified.

Use of the COPY REPLACING option requires that the "library-name" COBOL source image file be present, on disk, prior to compiling the source program containing the COPY REPLACING option. The use of this option will not cause alteration of the library file residing on disk.

In Option 2, literals contained in a library file cannot be replaced by literals, words or data-names.

In Option 2, if an integer is used for a word and it is the last entry in a replacing list, it must be followed by a blank and then a period. For example:

```
COPY REPLACING AAA BY HOURS,
BBB BY PAY-SCALE, 1234 BY 58b.
```

The COPY REPLACING option is exceptionally beneficial for conversion of generalized COBOL source language library routines into specific and well-named routines within a given program. For example, a generalized COBOL source language library routine may use the following data-names for their noted purposes:

Data-name	Purpose
AAA	Monthly hours worked per employee.
BBB	Employee pay-rate.
CCC	Employee social security number.
DDD	Employee income tax rate.
EEE	Employee year to date gross income.
FFF	Employee year to date net income.
GGG	Employee gross pay for month.
HHH	Employee gross pay for month.
.	.
.	.
.	.
1234	Specifies a GO exit from the routine.

A program calling upon the above generalized routine can replace the non-descript data-names with descriptive names as defined in the programs record description or WORKING-STORAGE area. For example:

```
COPY... REPLACING AAA BY HOURS-WORKED
COPY... REPLACING BBB BY RATE-OF-PAY
COPY... REPLACING CCC BY SOC-SEC-NR
COPY... REPLACING DDD BY INC-TAX-RATE
COPY... REPLACING EEE BY YR-TO-DATE-GROSS
COPY... REPLACING FFF BY YR-TO-DATE-NET
COPY... REPLACING GGG BY THIS-MONTHS-GROSS
COPY... REPLACING HHH BY THIS-MONTHS-NET
                          .
                          .
                          .
COPY... REPLACING 1234 BY WRITE-EMPLOYEE-DRAFT.
```

The specified source program data-names and exit points will be inserted into the library file routine at every occurrence of the assigned generalized names within the routine.

See appendix G for an example of a generalized Square Root Routine being specified by a COPY REPLACING option.

LIBRARY CREATION. A library file will be created only during a COBOL compilation each time that a source card is encountered containing an L in column 7 with a library-name, bounded by quotation marks starting in Field A of the same card. A library-file may contain up to a maximum of 20,000 card images.

Each library file in the source program will be terminated when a card containing an L in column 7 followed by all blanks or another library-name is encountered.

Library-names cannot start with a blank character or a dash (-).

Once a file has been created, it may be COPYed by other programs.

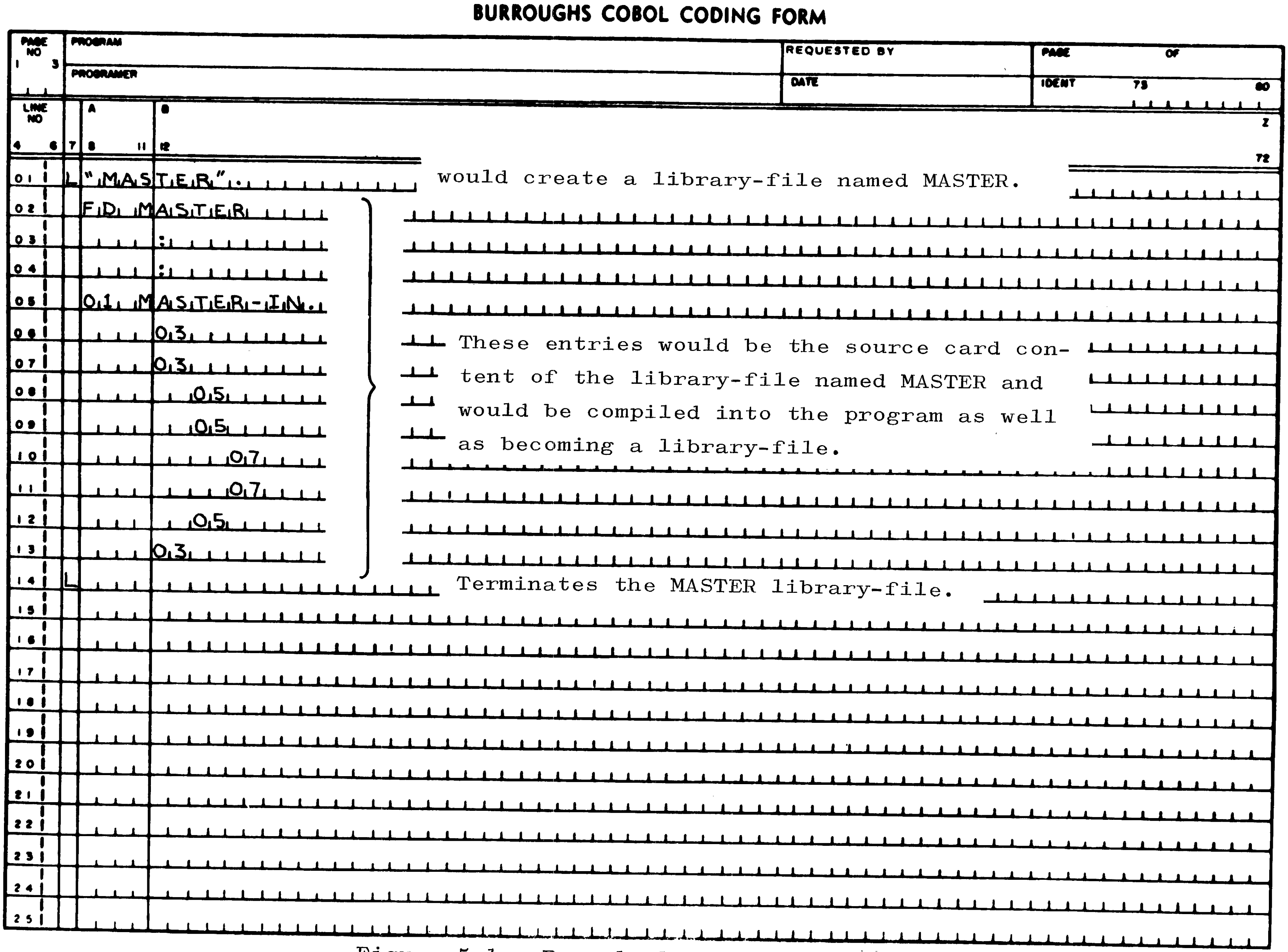
or the creating program in succeeding FD, 01, or procedure COPY statements.

The source data used to create an original library file will also be compiled into the object program at the point of appearance.

All assigned library-names must be unique to other library-names contained in the library to preserve the integrity of the COBOL library system.

Library files to be used with the COPY verb can be created by a user program which creates an unblocked card image file on disk.

Figure 5-1 through 5-8 provide examples of the COBOL COPY feature.



BURROUGHS COBOL CODING FORM

PAGE NO 1 3 | PROGRAM | REQUESTED BY | PAGE OF

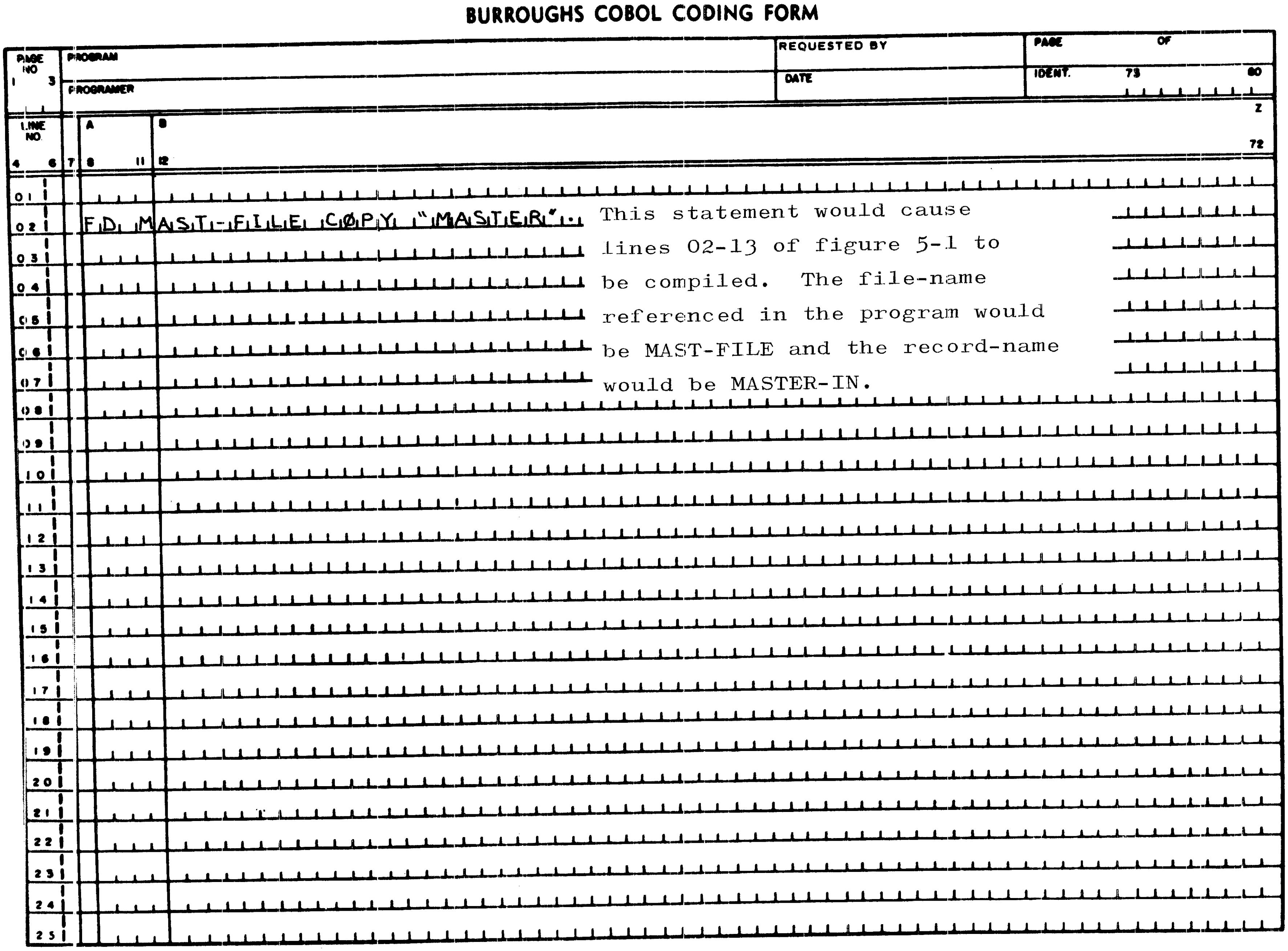
PROGRAMMER | DATE | IDENT 73 80

```
LINE NO  7  A         B                                                         Z
4     6     8     11  12                                                        72
01       L  "MASTER".            would create a library-file named MASTER.
02          FD MASTER
03              ;
04              ;
05          01 MASTER-IN.
06              03
07              03
08                 05
09                 05
10                    07
11                    07
12                 05
13              03
14       L                       Terminates the MASTER library-file.
15
16
17
18
19
20
21
22
23
24
25
```

Lines 02–13 (bracketed): These entries would be the source card content of the library-file named MASTER and would be compiled into the program as well as becoming a library-file.

Figure 5-1. Example 1 of COPY Coding

COPY continued



BURROUGHS COBOL CODING FORM

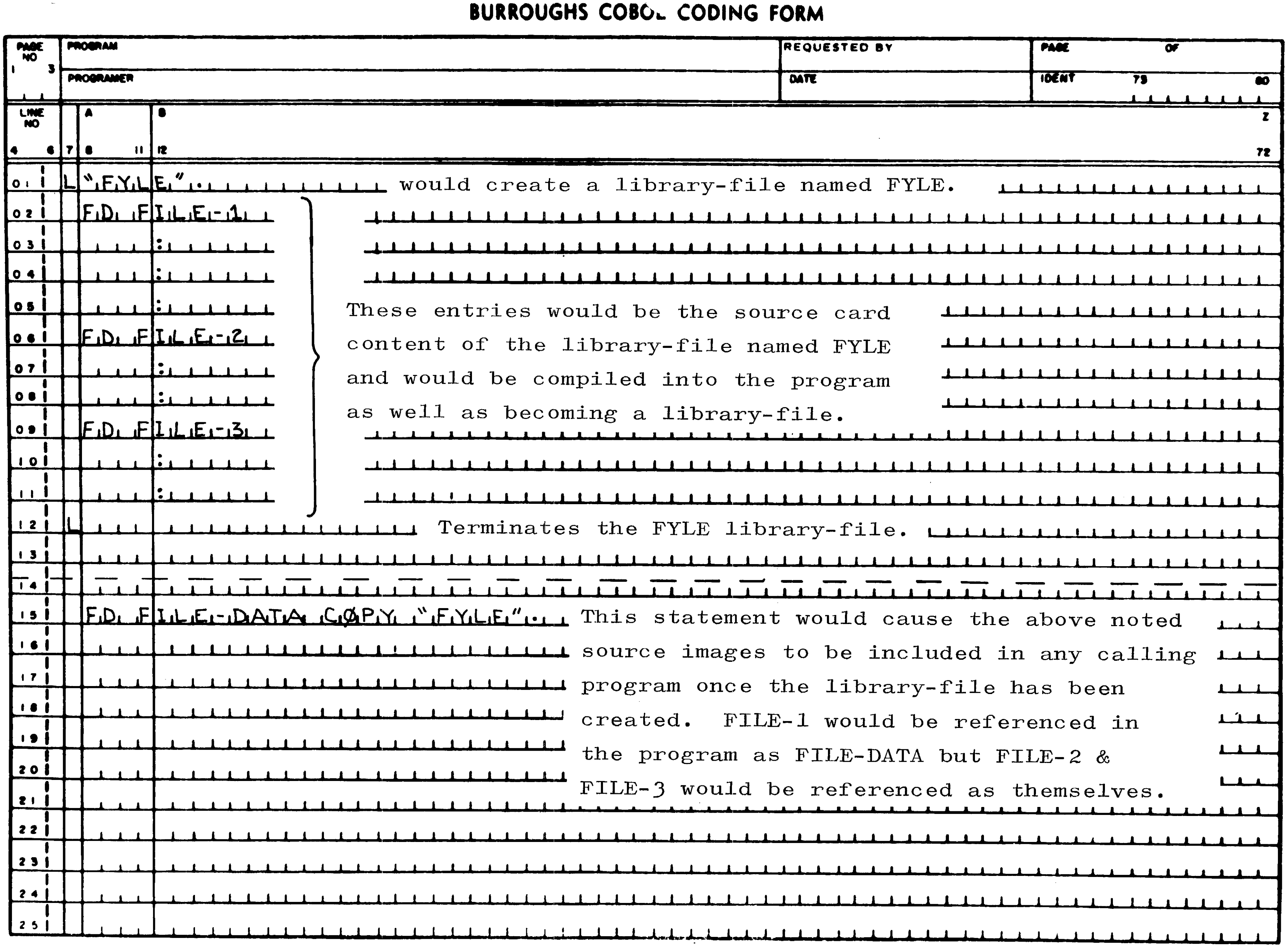
PAGE NO 1 3 | PROGRAM | REQUESTED BY | PAGE OF
PROGRAMMER | DATE | IDENT. 73 80

LINE NO. 4 6 | 7 | A 8 11 | B 12 | Z 72

```
02 FD MAST-FILE CØPY "MASTER".
```

This statement would cause lines 02-13 of figure 5-1 to be compiled. The file-name referenced in the program would be MAST-FILE and the record-name would be MASTER-IN.

Figure 5-2. Example 2 of COPY Coding



BURROUGHS COBOL CODING FORM

PAGE NO 1 3	PROGRAM	REQUESTED BY	PAGE OF
	PROGRAMER	DATE	IDENT 73 80

```
LINE  7  A       B                                                    Z
NO       8    11 12                                                   72
4   6
01    L "FYLE".                 would create a library-file named FYLE.
02       FD FILE-1
03          :
04          :
05          :               These entries would be the source card
06       FD FILE-2          content of the library-file named FYLE
07          :               and would be compiled into the program
08          :               as well as becoming a library-file.
09       FD FILE-3
10          :
11          :
12    L                          Terminates the FYLE library-file.
13
14
15       FD FILE-DATA COPY "FYLE".   This statement would cause the above noted
16                                   source images to be included in any calling
17                                   program once the library-file has been
18                                   created.  FILE-1 would be referenced in
19                                   the program as FILE-DATA but FILE-2 &
20                                   FILE-3 would be referenced as themselves.
21
22
23
24
25
```

Figure 5-3. Example 3 of COPY Coding

COPY
continued

[redacted]

BURROUGHS COBOL CODING FORM

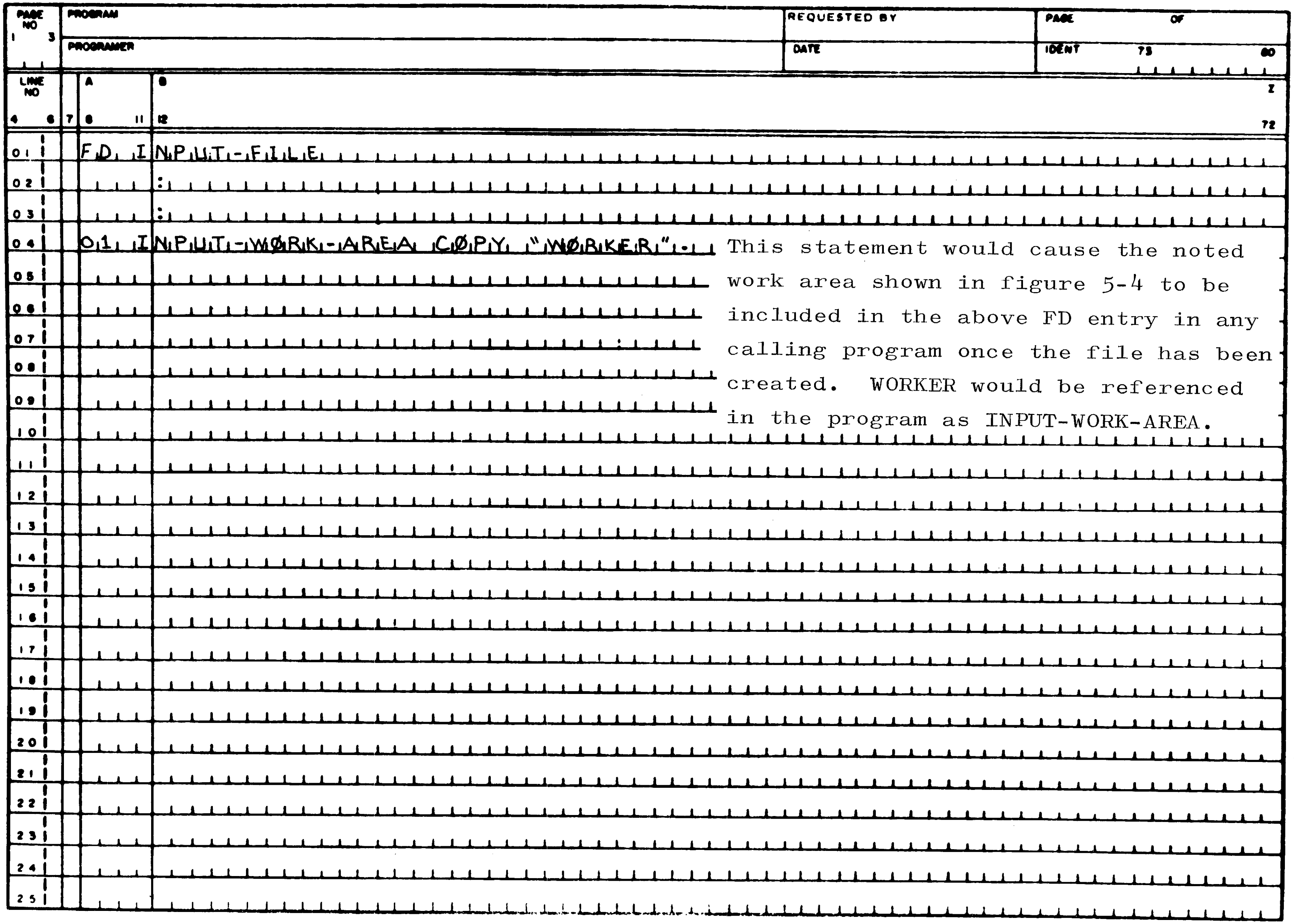
PAGE NO 1 3	PROGRAM	REQUESTED BY	PAGE OF
	PROGRAMER	DATE	IDENT 73 80

LINE NO 4 6 | 7 | A 8 11 | B 12 | Z 72

```
01 L"WØRKER".                      would create a library-file named WORKER.
02   01 WØRK-AREA.   }
03       03          }
04         05        }
05         07        }
06         09        }
07       03          }   These entries would be the source card con-
08         05        }   tent of the library-file named WORKER and
09             07    }   would be compiled into the program as well
10                09 }   as becoming a library-file.
11       03          }
12       03          }
13       05          }
14       07          }
15       09          }
16 L                               terminates the WORKER library-file.
17
18
19   WØRKING-STØRAGE SECTIØN.                This statement would cause the above
20   01 INPUT-WØRK-AREA CØPY "WØRKER".       noted source images to be included in
21                                           any calling program once the library
22                                           has been created.  WORKER would be
23                                           referenced in the program as INPUT-
24                                           WORK-AREA.
25
```

Figure 5-4. Example 4 of COPY Coding

BURROUGHS COBOL CODING FORM



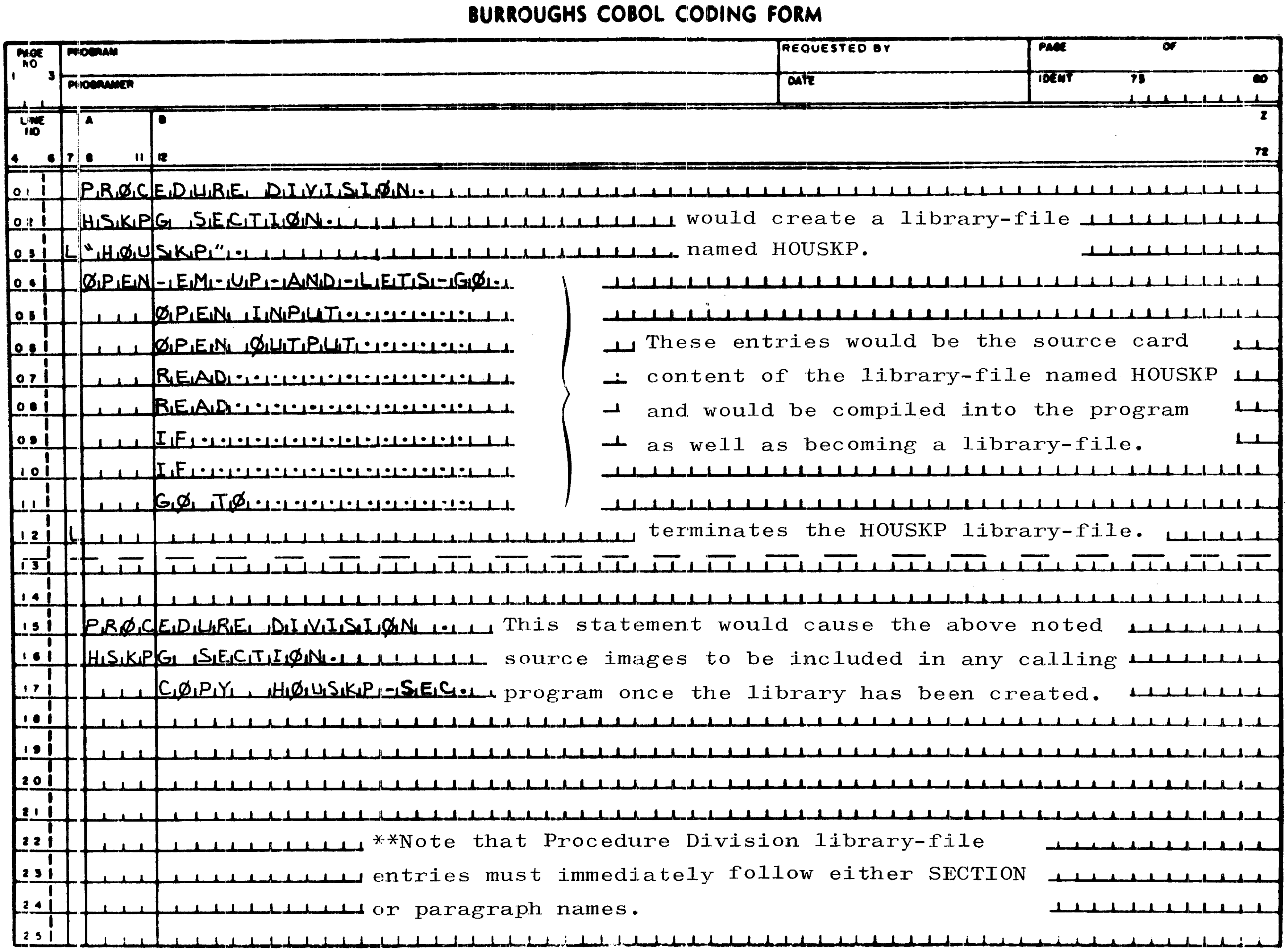
PAGE NO | PROGRAM | REQUESTED BY | PAGE OF
PROGRAMER | DATE | IDENT 73 80
LINE NO | A | B

```
       FD  INPUT-FILE
           :
           :
       01  INPUT-WØRK-AREA CØPY "WØRKER".
```

This statement would cause the noted work area shown in figure 5-4 to be included in the above FD entry in any calling program once the file has been created. WORKER would be referenced in the program as INPUT-WORK-AREA.

Figure 5-5. Example 5 of COPY Coding

COPY continued



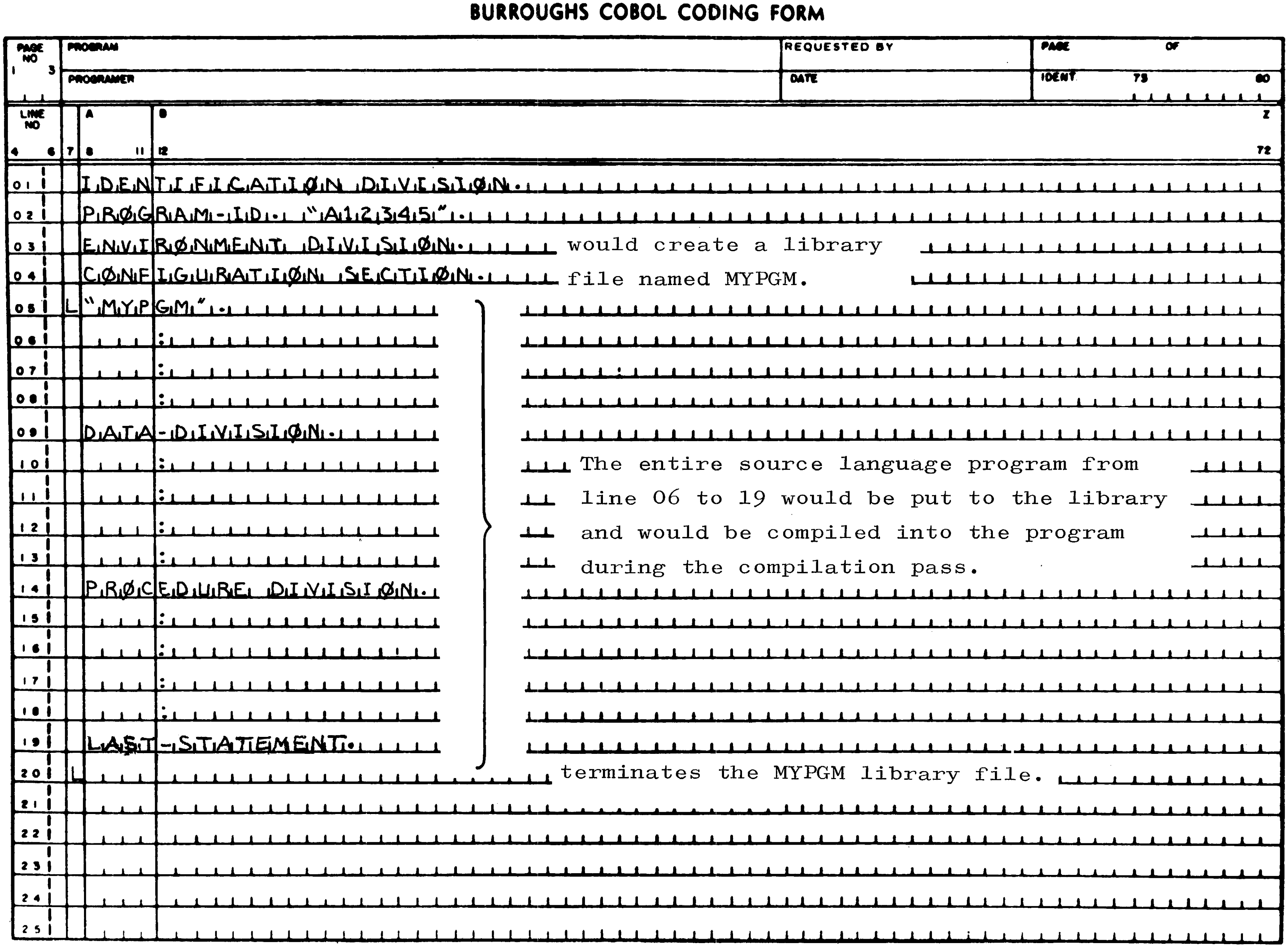
BURROUGHS COBOL CODING FORM

PAGE NO 1 3 | PROGRAM | REQUESTED BY | PAGE OF
PROGRAMMER | DATE | IDENT 73 80

LINE NO 4 6 | 7 | A 8 11 | B 12 | Z 72

```
PRØCEDURE DIVISIØN.
HSKPG SECTIØN.                      would create a library-file
L"HØUSKP".                          named HOUSKP.
ØPEN-EM-UP-AND-LETS-GØ.
    ØPEN INPUT........
    ØPEN ØUTPUT.......              These entries would be the source card
    READ..............              content of the library-file named HOUSKP
    READ..............              and would be compiled into the program
    IF................              as well as becoming a library-file.
    IF................
    GØ TØ.............
L                                   terminates the HOUSKP library-file.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

PRØCEDURE DIVISIØN.   This statement would cause the above noted
HSKPG SECTIØN.        source images to be included in any calling
    CØPY HØUSKP-SEC.  program once the library has been created.

**Note that Procedure Division library-file
entries must immediately follow either SECTION
or paragraph names.
```

Figure 5-6. Example of COPY Coding



BURROUGHS COBOL CODING FORM

PAGE NO 1 3 | PROGRAM | REQUESTED BY | PAGE OF
PROGRAMER | DATE | IDENT 73 80

LINE NO 4 6 | 7 | A 8 11 | B 12 | Z 72

```
01   IDENTIFICATIØN DIVISIØN.
02   PRØGRAM-ID. "A12345".
03   ENVIRØNMENT DIVISIØN.
04   CØNFIGURATIØN SECTIØN.
05  L"MYPGM".
06       :
07       :
08       :
09   DATA-DIVISIØN.
10       :
11       :
12       :
13       :
14   PRØCEDURE DIVISIØN.
15       :
16       :
17       :
18       :
19   LAST-STATEMENT.
20  L
21
22
23
24
25
```

Lines 03–04: would create a library file named MYPGM.

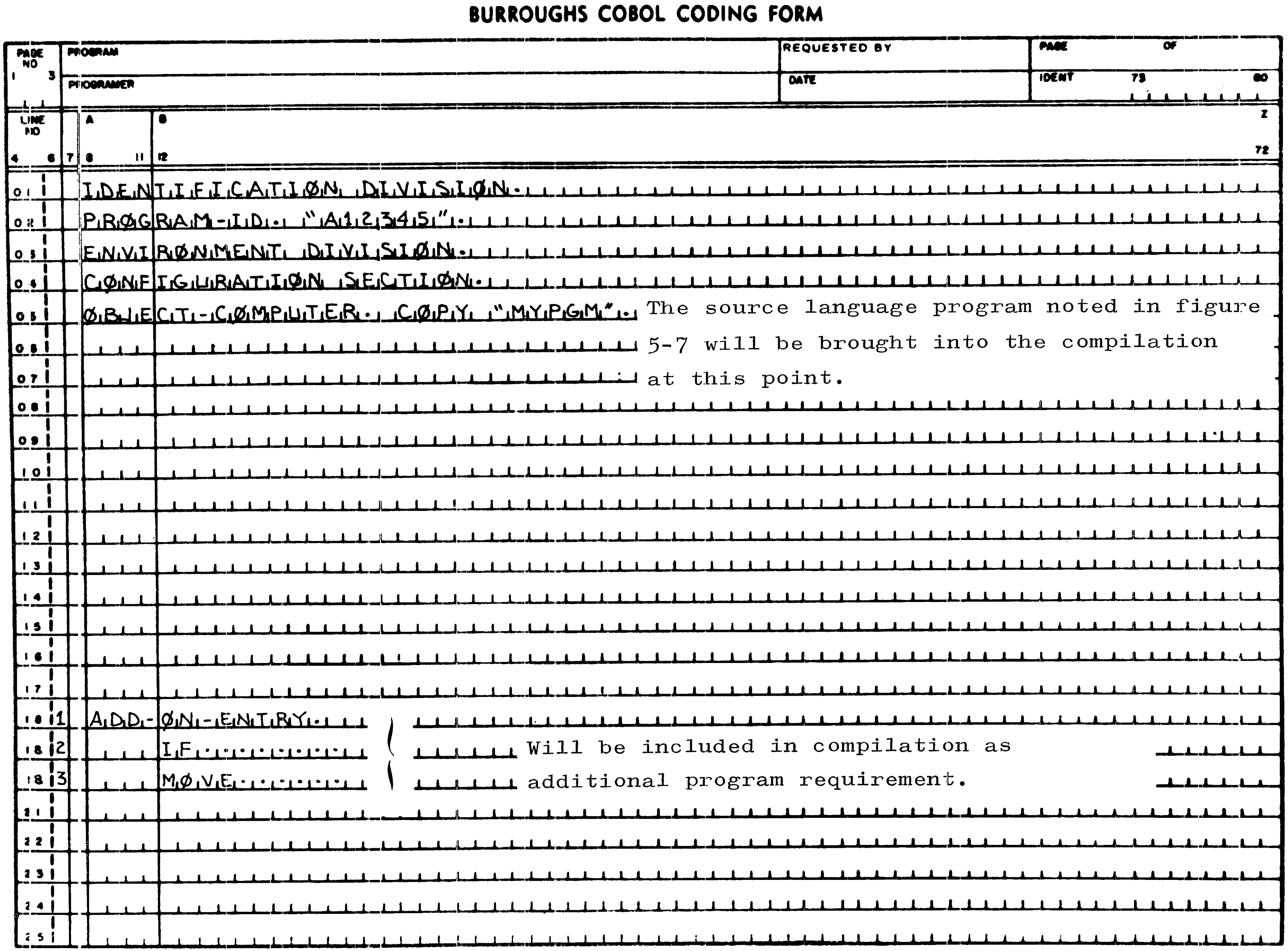
Lines 05–19: The entire source language program from line 06 to 19 would be put to the library and would be compiled into the program during the compilation pass.

Line 20: terminates the MYPGM library file.

Figure 5-7. Example 7 of COPY Coding

Revised 9/28/70
by PCN 103099-004

COPY
continued



BURROUGHS COBOL CODING FORM

PAGE NO 1 3	PROGRAM	REQUESTED BY	PAGE OF
	PROGRAMMER	DATE	IDENT 73 80

```
LINE
NO      A       B                                                    Z
4   6 7 8    11 12                                                   72
01      IDENTIFICATIØN DIVISIØN.
02      PRØGRAM-ID. "A12345".
03      ENVIRØNMENT DIVISIØN.
04      CØNFIGURATIØN SECTIØN.
05      ØBJECT-CØMPUTER. CØPY "MYPGM".
06
07
08
09
10
11
12
13
14
15
16
17
181     ADD-ØN-ENTRY.
182        IF..........
183        MØVE........
21
22
23
24
25
```

The source language program noted in figure 5-7 will be brought into the compilation at this point.

Will be included in compilation as additional program requirement.

Figure 5-8. Example 8 of COPY Coding

DISPLAY.

The function of this verb is to provide for the printing of low-volume data, error messages, and operator instructions on the console typewriter.

The construct of this verb is:

```
DISPLAY  { literal-1    }    [ { literal-2    }        ... ]
         { data-name-1  }      { data-name-2  }

    [ UPON  { SPO           } ]
            { mnemonic-name }
```

Each literal may be any figurative constant except ALL.

All special registers (DATE, TIME, etc.) may be DISPLAYed.

The DISPLAY statement causes the contents of each operand to be written on the supervisory printer (SPO) from the MCP SPO queue to ensure that a program is not operationally deterred while a message is printing.

If a figurative constant is specified as one of the operands, only a single character of the figurative constant is displayed.

The data-names may be subscripted and can be PICTUREd as COMPUTATIONAL or DISPLAY items.

An infinite amount of characters may be displayed with one statement. The compiler will supply automatic carriage returns and line feeds, as may be appropriate.

The DISPLAY series option will cause the literals or data-names to be printed on one line and, if required, the compiler will cause automatic carriage returns and line feeds for information extending to other lines of print. The compiler will format each line so that a partial word at an end of a line will not be printed on that line, and continued on the following line.

When mnemonic-name is used, it must appear in the SPECIAL-NAMES paragraph equated to the hardware-name SPO.

DIVIDE.

The function of this verb is to divide one numerical data-item into another and set the value of an item equal to the result.

The construct of this verb contains two options which are:

Option 1:

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

[ON SIZE ERROR any statement    [ { OTHERWISE }   statement ]]
                                [ { ELSE      }             ]
```

Option 2:

```
DIVIDE  [MOD]    { literal-1   }  { BY   }  { literal-2   }
                 { data-name-1 }  { INTO }  { data-name-2 }

   GIVING data-name-3 [ROUNDED]

  [REMAINDER data-name-4 [ROUNDED] ]

  [ON SIZE ERROR any statement   [ { OTHERWISE }   statement]]
                                 [ { ELSE      }            ]
```

Data-name-3 and data-name-4 of Option 2 may refer to a data item that contains editing symbols.

Each literal must be a numeric literal.

Division by zero is not permissible and, if executed, will result in a size error indication. This can be handled programmatically, either by doing a zero test prior to the division, or by the use of the SIZE ERROR clause. If SIZE ERROR is not written, an attempt to divide by zero will result in unpredictable results. Processing will continue.

All data-names must refer to elementary numeric items.

In Option 1, the value of the operand preceding the word INTO will be divided into the operand following INTO and the resulting quotient stored as the new value of the latter.

The use of the BY option will cause literal-1/data-name-1 to be divided <u>by</u> literal-2/data-name-2, whereas the INTO option will cause literal-1/data-name-1 to be divided <u>into</u> literal-2/data-name-2.

In Option 2, the resulting quotient will be stored as the new value of data-name-3. The value of the operands immediately to the left of the word GIVING will remain unchanged.

The ROUNDED option and ON SIZE ERROR clause and truncation are the same as discussed for the ADD statement (page 5-30).

The size of the operands is determined by the sum of the divisor and the quotient. The sum of the two cannot exceed 99 digits.

The use of the MOD option will cause the remainder to be placed in data-name-2 of Option 1 and data-name-3 of Option 2. The remainder will be carried to the same degree of accuracy as defined in the PICTURE of the quotient and all extra positions will be filled with zeros.

Literals cannot be used as dividends.

The use of the REMAINDER option will cause the remainder to be placed in data-name-4 and data-name-3 will contain the quotient, unless the MOD option is also included. If the MOD option is included, both data-name-3 and data-name-4 will contain the remainder.

END-OF-JOB.

The function of this verb is to notify the COBOL Compiler that all source statements within a program have been read.

The construct for this indicator is:

```
END-OF-JOB.
```

The END-OF-JOB statement is for documentation only but if used it must be the last source program card in a B 2500/B 3500 COBOL deck. It immediately precedes the MCP END Control Card.

ENTER.

This verb provides for the use of an alternate language.

The construct of this verb is:

$$\underline{\text{ENTER}} \quad \left\{ \begin{array}{l} \underline{\text{SYMBOLIC}} \\ \underline{\text{COBOL}} \end{array} \right\} .$$

ENTER COBOL is used at the point in the source program that ends the alternate language and where the programmer wants to continue with COBOL.

ENTER SYMBOLIC begins in column 12 or after and must be followed by a period. Each symbolic entry (instruction) may be followed by a period or the period may be omitted, however, the last instruction of a procedure must be followed by a period.

Procedure-names must start in columns 8-11 and must be followed by a period. Procedure-names may be used freely and referenced by branching instructions. It is permissible to reference procedure-names located outside of ENTER SYMBOLIC.

COBOL data-names may be used in the A, B, or C operands. No detailed syntax checking is performed other than data-name look-up, therefore, use of the symbolic language requires knowledge of the B 2500/B 3500 Assembler Language.

The symbolic operators can appear anywhere on the card after column 11. Any word or value in excess of the required number of operators will result in a syntax error. Literals (123, +123) of any size may be used in the A operand. If it is desired to carry an in-line literal the symbolic operator LIT is used. For example:

```
LIT 123456789
LIT @ABCDEF@
LIT "ABC567"
LIT @C1234@
```

The operating system requires that all in-line constants be character adjusted. In order to accomplish this, a zero may be added; thus, a literal 2 will generate as a literal 20. If it is necessary to use numbers such as this, it will be the programmer's responsibility to use leading zeros or at least be aware of this condition.

The following instructions may be used and require <u>two</u> operands, the first of which may be a literal. The length for AF and BF is taken from the data definition.

```
INC     SZU     MVA
MVN     DEC     CPN
MVR     CPA     SDU
SDE     SZE
```

The following instructions require <u>three</u> operands, the first of which may be a literal. The length for AF and BF is taken from data definitions.

```
ADD     SUB     MPY
DIV     FAD     FSU
FMP     FDV     AND
ORR     NOT     EDT
```

The following instructions require <u>one</u> operand that must be a procedure-name which appears either within or outside the ENTER statement.

```
NOP     LSS     EQL
LEQ     GTR     NEQ
GEQ     BUN     OFL
HBR
```

The following are exceptions to the above classification and must be used as shown.

Operator	Required No. of Operands	Description of Operands
BCT	1	Literal.
HBK	1	Literal.
NTR	2	Literal and procedure-name.
EXT	0	Only normal exits are allowed.
MVW MVC	3	Literal for number of words to be moved, A-ADDRESS, and B-ADDRESS.
TRN MVL	4	Literal size, A-ADDRESS, B-ADDRESS, and C-ADDRESS.
BZT BOT	3	Literal AF, literal BF, and A-ADDRESS.
SMF	1	Literal AF (0 or 1).
SEA	4	Literal BF, A-ADDRESS, B-ADDRESS, and C-ADDRESS (see pages 5-61 and 5-62 for more detailed explanation).

All the operators, literals (LIT), index register names (IX1, IX2, and IX3), data formats (INA, UNS, SGN, and DSP) and BAS (base) are reserved words within ENTER (only).

A symbolic operand has the following format:

$$\left\{\begin{array}{l}\text{data-name}\\ \text{file-name}\\ \underline{\text{IX1}}\\ \underline{\text{IX2}}\\ \underline{\text{IX3}}\\ \underline{\text{BAS}}\\ \underline{\text{TALLY}}\\ \underline{\text{SW1-SW8}}\end{array}\right\}\left[\ :\left\{\begin{array}{l}\underline{\text{INA}}\\ \underline{\text{UNS}}\\ \underline{\text{SGN}}\\ \underline{\text{DSP}}\end{array}\right\}:\left\{\begin{array}{c}+\\ -\end{array}\right\}\text{numeric-literal : numeric literal:}\left\{\begin{array}{l}\underline{\text{IX1}}\\ \underline{\text{IX2}}\\ \underline{\text{IX3}}\end{array}\right\}\right]$$

All options must be separated by a colon (:). The INA, UNS, SGN, and DSP specify data formats of indirect addressing, unsigned 4-bit, and DISPLAY 8-bit respectively.

The signed numeric literal option specifies that the starting address is to be incremented or decremented by the specified number of digits or characters, depending on the data format specified at that time.

The unsigned numeric literal specifies that the size of the field is to be changed to that value. The value should not exceed that allowed for the particular instruction or it will be truncated.

Index registers IX1, IX2, and IX3 used in conjunction with a data-name indicate that the index register bit for that operand is to contain that register setting. IX1, IX2, and IX3 may also be used as a data-name in which case the contents of the register will be affected.

No consistency checking is performed on any of these options; all, none, or any combination may appear. Duplicate appearances are not considered errors. The options are processed in the order in which they appear. Particular care must be exercised when changing data-format and also address on the same data-name. If the increment value precedes the UNS, etc., the address will be incremented according to the old format. Conversely, if it follows the format change, it will be incremented according to the changed format.

BAS (base) will provide the address 000000 with length specified as one, unsigned numeric. By using address and size modification, it is possible to reference any area within the program.

SEA (Search) requires four operands in the following format:

a. Increment - a numeric literal for BF.

b. A-ADDRESS - a literal or a data-name. Its size is used in the AF (size may be modified as previously described).

c. B-ADDRESS - data-name only. It must meet the requirements of SEARCH as in the B 2500/B 3500 Assemblers Reference Manual.

d. C-ADDRESS - any data-name. To set the C-ADDRESS as described in the Assemblers Reference Manual, use UNS, SGN, or DSP to set it to 00, 01, or 10 respectively.

The following is an example of the SEA (Search) command:

```
     SEA     05     AX     BX     BX:+500:SGN
```

COBOL type subscripting is not allowed in ENTER SYMBOLIC. It is the programmer's responsibility to set up any required indexing.

The use of a file-name as an operand will result in an address pointing to the start of the File Information Block (FIB) for that file, with 4-bit usage indicated. The size of the FIB is 200 digits, therefore, since the size field in an instruction is only two digits long, care must be exercised.

If a BUN or some other branching instruction is referencing a procedure-name in an overlayable segment, the programmer must do his own overlaying by setting up the segment dictionary or by being sure that the specific overlay is already in core. Another way is to code:

```
ENTER SYMBOLIC.
.......
ENTER COBOL.
     GO TO (overlay segment label).
ENTER SYMBOLIC.
.......
```

instead of using BUN. This will cause the COBOL Compiler to generate the overlay code.

If IX1, IX2, or IX3 are used in the ENTER SYMBOLIC statement, their values are not saved and, after re-entry of ENTER SYMBOLIC, the contents of these registers are unpredictable.

Modifiers may not be used with procedure-names (labels).

EXAMINE.

The function of this verb is to replace a specified character, and/or to count the number of occurrences of a particular character in a data item.

The construct of this verb is:

$$\underline{\text{EXAMINE}}\ \ \text{data-name}$$

$$\left\{\begin{array}{l} \underline{\text{TALLYING}} \left\{\begin{array}{l}\underline{\text{ALL}}\\ \underline{\text{LEADING}}\\ \underline{\text{UNTIL}}\ \underline{\text{FIRST}}\end{array}\right\} \left\{\begin{array}{l}\text{literal-1}\\ \text{data-name-1}\end{array}\right\} \left[\underline{\text{REPLACING}}\ \ \underline{\text{BY}} \left\{\begin{array}{l}\text{literal-2}\\ \text{data-name-2}\end{array}\right\}\right] \\ \\ \underline{\text{REPLACING}} \left\{\begin{array}{l}\underline{\text{ALL}}\\ \underline{\text{LEADING}}\\ \left[\underline{\text{UNTIL}}\right]\ \underline{\text{FIRST}}\end{array}\right\} \left\{\begin{array}{l}\text{literal-3}\\ \text{data-name-3}\end{array}\right\}\ \ \underline{\text{BY}} \left\{\begin{array}{l}\text{literal-4}\\ \text{data-name-4}\end{array}\right\} \end{array}\right\}$$

The description of data-name must be such that USAGE is DISPLAY explicitly or implicitly.

Each literal used in an EXAMINE statement must consist of a single DISPLAY character. Figurative constants will automatically represent a single DISPLAY character.

Examination proceeds as follows:

a. For items that are not numeric (4-bit), examination starts at the left-most character and proceeds to the right. Each 8-bit character in the item specified by the data-name is examined in turn. Any reference to the first character means the left-most character.

b. If an item referenced by the EXAMINE verb is numeric, it must consist of numeric (8-bit) characters and may possess an operational sign. Examination starts at the left-most character (excluding the sign) and proceeds to the right. Each character except the sign is examined in turn. Regardless of where the sign is physically located, it is completely ignored by the EXAMINE verb. Any reference to the first character means the left-most numeric character.

The TALLYING option creates an integral count (i.e., a tally) which replaces the value of a special register called TALLY. The count represents the number of:

a. Occurrences of literal-1 or data-name-1 when the ALL option is used.

b. Occurrences of literal-1 or data-name-1 prior to encountering a character other than literal-1 or data-name-1 when the LEADING option is used.

c. Characters not equal to literal-1 or data-name-1 encountered before the first occurrence of literal-1 or data-name-1 when the UNTIL FIRST option is used.

When either of the REPLACING options is used (i.e., with or without TALLYING) the replacement rules are as follows:

a. When the ALL option is used, then literal-2 or data-name-2 or literal-4 or data-name-4 is substituted for each occurrence of literal-1 or data-name-1 or literal-3 or data-name-3.

b. When the LEADING option is used, the substitution of literal-2 or data-name-2 or literal-4 or data-name-4 terminates as soon as a character other than literal-1 or data-name-1 or literal-3 or data-name-3 or the right-hand boundary of the data item is encountered.

c. When the UNTIL FIRST option is used, the substitution of literal-2 or data-name-2 or literal-4 or data-name-4 terminates as soon as literal-1 or data-name-1 or literal-3 or data-name-3 or the right-hand boundary of the data item is encountered.

d. When the FIRST option is used, the first occurrence of literal-3 or data-name-3 is replaced by literal-4 or data-name-4.

The field called TALLY is a 5-digit field provided by the compiler. Its usage is COMPUTATIONAL and will be reset to zero automatically when the EXAMINE...TALLY option is encountered.

EXIT.

The function of this verb is to provide a terminating point for a PERFORM loop, whenever required.

The construct of this verb is:

EXIT.

If the EXIT statement is used, it must be preceded by a paragraph-name and appear as a single one-word paragraph. EXIT is documentational only, but if used, must follow the rules of COBOL.

The EXIT is normally used in conjunction with conditional statements contained in procedures referenced by a PERFORM statement. This allows branch paths within the procedures to rejoin at a common return point.

If control reaches an EXIT paragraph and no associated PERFORM or USE statement is active, control passes through the EXIT point to the first sentence of the next paragraph and is treated for all intents and purposes as a NOP (No Operation).

FILL.

The function of this verb is to pass data from one program to another when both programs are operating in the same multiprogramming mix.

The construct of this verb has two options which are:

Option 1:

```
FILL data-name-1 INTO  { non-numeric literal-1 }
                       { data-name-2           }

 [PROCEED TO paragraph-name]
```

Option 2:

```
FILL data-name-3 FROM  { non-numeric literal-2 }
                       { data-name-4           }

 [PROCEED TO paragraph-name]
```

The MCP Core to Core (CRCR) option must be set "ON" when an object program containing the FILL verb is being operated under the control of a standard version of the MCP.

Option 1 is the data-sending construct whereby a program using this statement can converse from a self-contained data-name, with another operating program in the same multiprogramming mix. The size of data-name-1 is restricted only by the amount of memory required by the programs themselves. Data-name-1 must be declared as alphanumeric in a DISPLAY mode. Data-name-2 must be declared as a PICTURE X(6) which specifies the program-identifier of the receiving program as reflected in the MCP Program Directory. The receiving program must be in the MCP Mix. If the non-numeric literal-1 is "bbbbbb" (blank), it specifies that any number of receiving programs are to become eligible for the transmission of data. The PROCEED TO clause, when specified, will cause a branch to a paragraph-name when there is no receiving program ready to receive a transmission. If this clause is not used, the program will wait until the FILL has been completed, before proceeding to the next instruction.

Option 2 is the data-receiving construct whereby a program using this statement can receive data from a multiprogramming sending program (data-name-4) into a self contained data-name-3. Data-name-3 must be declared as alphanumeric in a DISPLAY mode. The sending program must be in the MCP Mix. If the non-numeric-literal-2 is "bbbbbb" (blank), it specifies that any number of sending programs are to become eligible for the transmission of data. The PROCEED TO clause, when specified, will cause a branch to a paragraph-name if the sending program is not ready to transmit.

Data-name-2 and data-name-4 may not be subscripted or indexed.

Reference should be made to the FILL verb located in section 6 (Data Communications) of this manual. That construct, when specified, requires the presence of a version of the Data Communications MCP at object program execution time.

GO.

The function of this verb is to provide a means of breaking out of the sequential, sentence by sentence, execution of code, and to permit continuation at some other location indicated by the procedure-name(s).

The construct of this verb has two options which are:

Option 1:

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

Option 2:

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

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

In Option 2, GO... DEPENDING... may specify up to 999 procedure-names in a single statement.

In Option 2, the data-name in the format following the words DEPENDING ON must be a numeric elementary item described without any positions to the right of the assumed decimal point. Furthermore, the value must be positive in order to pass control to the procedure-names specified. Control will be transferred to procedure-name-1 if the value of the identifier is 1, to procedure-name-2 if the value is 2, etc. If the value of the identifier is anything other than a positive integer, or if its value is zero, or its value is higher than the number of procedure-names specified, control will be passed to the next statement in normal sequence. For example:

GO TO MFG, RE-SALE, STOCK, DEPENDING ON S-0.

Value of S-0	GO TO procedure-name
-1	next sentence
0	next sentence
1	MFG
2	RE-SALE
3	STOCK
4	next sentence

Whenever a GO statement (represented by Option 1) is executed, control is unconditionally transferred to a procedure-name, or to another procedure-name if the GO statement has been changed by an ALTER statement.

A GO statement is unrestricted as to where it branches to in a segmented program. It can call upon any segment (fixed or overlayable) at either section level or paragraph levels nested to any depth within a section.

When, in Option 1, the GO statement is referred to by an ALTER statement, the following rules apply regardless of whether or not procedure-name is specified:

a. The GO statement must have a paragraph-name.

b. The GO statement must be the only statement in the paragraph.

c. If the procedure-name is omitted, and if the GO statement is not referenced by an ALTER statement prior to the first execution of the GO statement, the MCP will terminate the job and cause an error message reflecting an invalid address.

If a GO statement represented by Option 1 appears in an imperative statement, it must appear as the only or the last statement in a sequence of imperative statements.

IF.

The function of this verb is to control the sequence of commands to be executed depending on either a condition, the class status of a field, or the relative value of two quantities. The purpose of a condition is to cause the object program to select between alternate paths depending on the passing or failing of the test.

The conditions are subdivided into six major categories which are:

a. Simple conditional tests.
b. Conditional statements.
c. Relation tests.
d. Relation value tests.
e. Class tests.
f. Conditional variable tests.

SIMPLE CONDITIONAL TESTS. The simple conditional tests are contained in option 1.

Option 1:

```
IF  condition-1  statement-1
```

CONDITIONAL STATEMENTS . A conditional statement specifies that the truth value of "yes" in a given condition is to be determined and that subsequent action of the object program is contingent upon the resultant value. READ and WRITE statements which specify an INVALID KEY option, or arithmetic statements (ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT) which specify a SIZE ERROR option are considered as being conditional.

In Option 2, statement-1 or statement-2 can be either imperative or conditional. If conditional, it can in-turn contain conditional nested statements to an arbitrary depth.

See Section 8, COBOL Compiler Option Card, for information pertaining to the generation of the SEARCH operator into an object program for conditional statements.

Option 2:

```
IF  condition  { statement-1   } [{ OTHERWISE } { statement-2   }]
               { NEXT SENTENCE } [{ ELSE      } { NEXT SENTENCE }]
```

RELATION TESTS. A relation test involves a comparison of two operands; either of which can be a data-name, a literal, or a formula. The comparison of two literals is not permitted. Comparison of elementary numeric items is permitted regardless of their individual USAGEs. All other comparisons require that the USAGE of the items being compared be the same. Group numeric items are defined to be alphanumeric. It is not permissable to compare an index-data-name against a literal or a data-name. The format of relation tests is shown in Option 3.

Option 3:

```
                                                  { =            }
    { literal-1                }                  { >            }
IF  { data-name-1              }   IS [NOT]       { <            }
    { arithmetic expression-1  }                  { EQUAL TO     }
                                                  { LESS THAN    }
                                                  { GREATER THAN }
                                                  { EQUALS       }
    { literal-2                }
    { data-name-2              }
    { arithmetic expression-2  }
```

RELATIVE VALUE TESTS. The relative value test is an alternate way of stating a comparison of the value zero with a formula, or with data-name. An item or formula is POSITIVE only if its value is greater than zero. An item or formula is NEGATIVE only if its value is less than zero. The value zero is considered neither POSITIVE nor NEGATIVE. This form of comparison with zero is not considered a relational test. The format of relative value tests is as follows:

Option 4:

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

CLASS TESTS. The class test is used to determine whether the contents of the data-name is made up entirely of NUMERIC or ALPHABETIC DISPLAY characters. For example:

"JOHN DOE" is ALPHABETIC	[PC X(8)]
"R. JOHN DOE" is not ALPHABETIC	[PC X(11)]
"37373" is NUMERIC	[PC 9(5)]
"234" is NUMERIC	[PC 9(3)]
"-37452" is NUMERIC	[PC S9(5)]
"685.57" is not NUMERIC	[PC X(6)]

The format of the class test is as follows:

Option 5:

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

CONDITIONAL VARIABLE TESTS. A conditional variable test is one in which an item is tested to determine whether or not the value associated with a condition-name is present. The rules for comparing a conditional variable with a conditional value are the same as those for relation tests. The format for a conditional variable test is:

Option 6:

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

Not Logic.

The statement:

```
IF A IS NOT EQUAL TO B OR C OR D, GO TO paragraph-name-1
             ELSE GO TO paragraph-name-2.
```

a. Condition-1. If A is not equal to B, control will transfer immediately to paragraph-name-1.

b. Condition-2. If A equals B, a test of C for inequality is set up. If C is unequal, control transfers immediately to paragraph-name-1; but if C is also equal, a test of D for inequality is set up. If D is unequal, control transfers immediately to paragraph-name-1; but if D is also equal, program control transfers immediately to paragraph-name-2.

c. Conclusion. The above explanation reflects that a test of field A versus the fields B OR C OR D for unequal status in all fields during one operation is an impossibility when using NOT/OR logic. The first data field reflecting inequality will cause a branch to be executed to paragraph-name-1.

d. In the above example, had AND logic been applied, the tests would have been accomplished in the very same manner.

MOVE.

The function of this verb is to transfer data from one area of memory to one or more data areas (receiving fields). The data will be automatically edited or adjusted as to the applicable PICTURE and USAGE clauses.

The construct of this verb is:

Option 1:

$$\underline{\text{MOVE}} \quad \left\{ \begin{matrix} \text{literal-1} \\ \text{data-name-1} \end{matrix} \right\} \quad \underline{\text{TO}} \quad \text{data-name-2} \; [\text{data-name-3...}]$$

Option 2:

$$\underline{\text{MOVE}} \quad \left\{ \begin{matrix} \underline{\text{CORR}} \\ \underline{\text{CORRESPONDING}} \end{matrix} \right\} \quad \text{data-name-1} \; \underline{\text{TO}} \; \text{data-name-2}$$

The MOVE statement without the CORR or CORRESPONDING option may not be used to MOVE a group item if editing or conversion of elementary items is desired. To do this, either the CORR or CORRESPONDING option must be used, or each elementary item must be moved individually. CORR is an acceptable substitute for CORRESPONDING.

If the CORR or CORRESPONDING option is used, selected sending fields are MOVEd to selected receiving fields. Data-name-1 and data-name-2 must be group items. A pair of data items, one from data-name-1 and one from data-name-2, correspond if the data items in both have the same name and the same qualification up to, but not including, data-name-1 and data-name-2. At least one of the data items of both data-name-1 and data-name-2 must be an elementary item. Neither data-name-1 nor data-name-2 may be data items with levels 66, 77, or 88. Each data item which is subordinate to data-name-1 and data-name-2, and which contains a RENAMES clause, is ignored. Furthermore, a data item that is subordinate to data-name-1 and data-name-2 and contains a REDEFINES or OCCURS clause is ignored. However, data-name-1 and data-name-2 may have REDEFINES or OCCURS clauses or be subordinate to data items with these clauses.

The CORR or CORRESPONDING option generates the following:

a. Elementary to elementary.
b. Elementary to group.
c. Group to elementary MOVEs within the two record descriptions.

Any MOVE in which the sending field and receiving items are elementary items is an elementary MOVE. Every elementary item belongs to one of the following categories: alphabetic, numeric, alphanumeric, numeric edited, or alphanumeric edited. These categories are discussed in PICTURE. Numeric literals belong to the numeric (4-bit) category, and non-numeric literals belong to the alphanumeric (byte) category. The following rules apply to an elementary MOVE between these categories:

a. In a MOVE of ALPHABETIC information to numeric field, the results will be unpredictable.

b. A numeric edited, alphanumeric edited, or alphabetic data item must not be MOVEd to a numeric or numeric edited data item.

c. A numeric or numeric edited data item must not be MOVEd to an alphabetic item.

d. A numeric item whose implicit decimal point is not immediately to the right of the least-significant digit must not be MOVEd to an alphanumeric or alphanumeric edited data item.

e. All other elementary moves are legal and are performed according to the rules outlined below:

1) An alphanumeric to alphanumeric elementary MOVE passes data constructed of bytes to a receiving

field constructed of bytes.

2) When an alphanumeric edited, alphanumeric, or alphabetic item is a receiving item, left justification occurs and any necessary space filling takes place to the right. If the length of the sending item is greater than the length of the receiving item, the right-most characters are truncated (see JUSTIFIED for the inverse procedure).

3) When a numeric or numeric edited item is a receiving item, alignment by decimal point and any necessary zero filling takes place except where zeros are replaced because of editing requirements. If the receiving item has no operational sign, the absolute value of the sending item is used. If the sending item has more digits to the left or right of the decimal point than the receiving item can contain, the excess digits are truncated. If the sending item contains non-numeric characters, the following actions occur:

   a) Zone bits will be stripped if the receiving field is COMP.

   b) Zone bits may be replaced with the numeric stick if the receiving field is DISPLAY.

4) Any necessary conversion of data from one form of internal representation to another takes place during the MOVE, along with any specified editing in the receiving item.

NOTE

Alphabetic or alphanumeric fields which are word aligned and contain an even number of bytes in length may create more efficient object code than those not synchronized.

NOTE (cont)

This is due to the fact that the compiler may generate the operation code for MOVE WORDS instead of the normal MOVE ALPHANUMERIC. As much as one-half of the normal operational time can be saved on selecting routines where the program loops through many iterations of data movement operations.

Any MOVE in which one or both operands is a group item, regardless of USAGE, is treated exactly as if it were an alphanumeric to alphanumeric elementary MOVE. There will be no conversion of data from one form of internal representation to another unless one of the fields is an elementary COMPUTATIONAL item. Group COMPUTATIONAL receiving fields are treated as if they are alphanumerically declared.

The following are examples of the MOVE statement:

a. The following examples show truncation of digits in moving numeric information.

Receiving Field Picture	9999	9900	9009	990099	0099	99/99
Value	1234	1234	1234	1234	1234	1234
Receiving Field	1234	3400	3004	120034	0034	12/34
Warning Message	No	Yes	Yes	No	Yes	No

b. The following examples show alignment of decimal points in moving numeric data. The symbol V denotes the assumed decimal point given by item description PICTURE clause, but which is not physically present.

Sending Field	Receiving Field	
Before and After	Before	After
123V45	0020V20	0123V45
123V45	002V020	123V450
123V45	00202V0	00123V4

c. The following example shows results of MOVE ALL statements. The use of a figurative constant ZERO in a MOVE statement will result in the entire DISPLAY or COMPUTATIONAL elementary receiving field being composed of zeros, with or without the use of the reserved word ALL. Therefore, MOVE ALL ZEROS, MOVE ZEROS, and MOVE ALL 0 are synonymous and will cause the DISPLAY or COMPUTATIONAL elementary receiving field to be composed of 8-bit or 4-bit zeros respectively.

	Five Position Receiving Field After Execution	
Statement	COMPUTATIONAL	DISPLAY
MOVE ALL 9 (or "9")	99999	F9F9F9F9F9
MOVE ALL 57	57575	F5F7F5F7F5
MOVE ALL 057	05705	F0F5F7F0F5
MOVE ALL "ABC"	*	C1C2C3C1C2
MOVE ALL ZEROS	00000	F0F0F0F0F0
MOVE ALL 0	00000	F0F0F0F0F0

* Unpredictable

MULTIPLY

MULTIPLY.

The function of this verb is to multiply two operands and store the results in the last-named field (which must be a numeric data-name).

The construct of this verb is:

```
MULTIPLY   { literal-1   }   BY   { literal-2   }
           { data-name-1 }        { data-name-2 }

[GIVING data-name-3]  [ROUNDED]  [ON SIZE ERROR  any

statement    [{ OTHERWISE }    statement ]]
              { ELSE      }
```

All rules specified under the ADD statement regarding the presence of editing symbols in operands, the ON SIZE ERROR option, the ROUNDED option, the GIVING option, truncation, and the editing results apply to the MULTIPLY statement, except the maximum operand size is 99 digits for the sum of two operands.

The data-names must be elementary item references. If GIVING is used, the data description of data-name-3 may contain editing symbols. In all other cases, the data-names used must refer to numeric items only.

If the GIVING option is used, the result of the multiplication replaces the contents of data-name-3, otherwise, it replaces the contents of data-name-2. If GIVING is not used, literal-2 is not permitted, i.e., data-name-2 must appear.

NOTE.

The function of this verb is to allow the programmer to write explanatory statements in his program which are to be produced on the source program listing for documentational clarity.

The construct of this verb is:

Option 1 - Paragraph NOTE:

```
label.  NOTE any comment.
```

Option 2 - Paragraph NOTE:

```
NOTE.  any comment.
```

Option 3 - Sentence NOTE:

```
NOTE any comment.
```

Any combination of the characters from the allowable character set may be included in the character string of a NOTE statement.

If a NOTE sentence is the first sentence of a paragraph, the entire paragraph is considered to be commentary. Either Option 1 or Option 2 may be used as NOTE statements on a paragraph level.

If a NOTE statement appears as other than the first sentence of a paragraph, only the sentence constitutes a commentary. The first

NOTE
continued

period after encountering the word NOTE will cause the compiler to resume compilation unless the new sentence commences with the word NOTE.

Refer to page 7-3 of Section 7, CONTINUATION INDICATOR, for an explanation of notes (* or / in column 7) appearing anywhere within the source program.

OPEN.

The function of this verb is to initiate the processing of both input and output files. The MCP performs checking or writing, or both, of labels and other input-output operations.

The construct of this verb is:

```
OPEN

[                    [ ( WITH LOCK [ACCESS] ) ]                     ]
[ INPUT file-name-1  [ < REVERSED           > ]  [file-name-2...]   ]
[                    [ ( WITH NO REWIND     ) ]                     ]

[ OUTPUT file-name-3 [WITH NO REWIND] [file-name-4 ...] ]

[ ( INPUT-OUTPUT )                    [                      ]
[ < I-O          >    file-name-5     [ WITH LOCK [ACCESS]   ]
                                      

     [file-name-6...] ]

[ O-I file-name-7 [file-name-8 ...] ]
```

File-names must not be those defined as being SORT files.

At least one of the options must be specified before a file can be read. A statement of OPEN INPUT.......OUTPUT.......I-O........... O-I.......can appear in one source language card. Continuation of source card lines is allowed.

The I-O, INPUT-OUTPUT and O-I options pertain to disk storage files. The I-O and INPUT-OUTPUT options may be used by data communication remote devices.

The OPEN statement must be executed prior to the first SEEK, READ, or WRITE statement for that file.

A second OPEN statement for a file cannot be executed prior to the execution of a CLOSE statement for that file.

The OPEN statement does not obtain or release the first data record. A READ or WRITE statement must be executed to obtain or release, respectively, the first data record.

When checking or writing the first label, the user's beginning label subroutine is executed if it is specified by a USE statement.

The REVERSED and the NO REWIND options can only be used with SEQUENTIAL, single reel, tape files.

If the peripheral ASSIGNed to the file permits rewind action, the following rules apply:

a. When neither the *REVERSED* nor the NO REWIND option is specified, execution of the OPEN statement for the file will cause the file to be positioned ready to read the first data-record.

b. When *either* the REVERSED or the NO REWIND option is specified, execution of the OPEN statement does not cause the file to be positioned. When the REVERSED option is specified, the file must be positioned at its physical *end*. When the NO REWIND option is specified, the file must be positioned at its physical *beginning*.

c. When the NO REWIND option is specified, it applies only to sequential, single reel files stored on magnetic tape units.

When the REVERSED option is specified, the subsequent READ statements for the file make the data-records available in reverse record order starting with the last record. Each record will be read into its record-area and will appear as if it had been read from a forward moving file.

If an input file is designated with the OPTIONAL clause in the File-Control paragraph of the ENVIRONMENT DIVISION, the object program causes an interrogation to the MCP for the presence or absence of a pertinent magnetic tape file. If this file is not present, the first READ statement for this file causes the imperative statement in the AT END clause to be executed.

The I-O or INPUT-OUTPUT option permits the OPENing of a disk file for input and/or output operations. This option demands the existence of the file to be on the disk and cannot be used if the file is being initially created. That is, the file to be OPENed must be present in the MCP Disk Directory.

When the I-O or INPUT-OUTPUT option is used, the MCP immediately checks the MCP Disk Directory to see if the file-name is present. The system operator will be notified in its absence, and the file can then be loaded if it is available or the program can be DSed (Discontinued). If the decision is to load the file, the operator does so and then notifies the MCP to proceed with the program by a mix-index OK message.

The O-I option is identical to OPEN I-O with the exception being that the file is assumed to be a new file to the Disk Directory. The OPEN O-I option will shortcut the usual method of initially creating I-O work files within a program, e.g., OPEN OUTPUT, write record(s), CLOSE WITH RELEASE, OPEN I-O, etc. The O-I option does not, nor was it intended to, replace the OPEN I-O option, since the use of OPEN O-I assumes that a new file is to be created each time.

When processing mass storage files for which the access mode is sequential, the OPEN statement supplies the initial address of the first record to be accessed.

The contents of the data-names specified in the FILE-LIMIT clause of the File-Control paragraph (at the time the file is OPENed) is used for all checking operation while that file is OPEN. The FILE-LIMIT clause is dynamic only to this extent.

When an OPEN OUTPUT statement is executed for a magnetic tape file, the MCP searches the assignment table for an available scratch tape, writes the label as specified by the program, and executes any USE declaratives for the file. If no scratch tape is available, a message to the operator is typed and the program is suspended until the operator mounts one, or one becomes available due to the termination of a multiprocessing program.

OPENing of subsequent reels of multi-reel tape files is handled automatically by the MCP and requires no special consideration from the programmer.

OPEN WITH LOCK on a permanent disk file (contained in the Disk Directory) denies the OPENing of that file to all other operating programs in the Mix. If another program attempts to OPEN INPUT of a file already OPENed with LOCK, the MCP control message ** LOCKED FILE file-name program-name = mix-index will appear on the SPO. The systems operator may continue the operation of the program by performing the control message mix-index OK only after the program LOCKing the file has performed a CLOSE on it.

OPEN WITH LOCK ACCESS on a permanent file (contained in the Disk Directory) allows the subsequent OPENing of that file by other operating programs in the Mix, as long as they only use the OPEN INPUT convention.

PERFORM.

The function of this verb is to depart from the normal sequence of execution in order to execute one or more procedures, either a specified number of times or until a specified condition is satisfied. Following this departure, control is automatically returned to the normal sequence.

The construct of this verb has four options which are:

Option 1:

$$\underline{\text{PERFORM}}\ \text{procedure-name-1} \left[\begin{Bmatrix} \underline{\text{THRU}} \\ \underline{\text{THROUGH}} \end{Bmatrix} \text{procedure-name-2} \right]$$

Option 2:

$$\underline{\text{PERFORM}}\ \text{procedure-name-1} \left[\begin{Bmatrix} \underline{\text{THRU}} \\ \underline{\text{THROUGH}} \end{Bmatrix} \text{procedure-name-2} \right]$$

$$\begin{Bmatrix} \text{integer-1} \\ \text{data-name-1} \end{Bmatrix} \quad \underline{\text{TIMES}}$$

Option 3:

$$\underline{\text{PERFORM}}\ \text{procedure-name-1} \left[\begin{Bmatrix} \underline{\text{THRU}} \\ \underline{\text{THROUGH}} \end{Bmatrix} \text{procedure-name-2} \right]$$

$$\underline{\text{UNTIL}}\quad \text{condition-1}$$

Option 4:

```
PERFORM procedure-name-1 [ { THRU    }  procedure-name-2 ]
                           { THROUGH }

VARYING { index-name-1 }  FROM { index-name-2       }  BY
        { data-name-1  }       { data-name-2        }
                               { numeric-literal-1  }

{ data-name-3       }  UNTIL condition-1  [ AFTER { index-name-3 }
{ numeric-literal-2 }                             { data-name-4  }

FROM   { index-name-4      }  BY  { data-name-6       }
       { data-name-5       }      { numeric-literal-4 }
       { numeric-literal-3 }

UNTIL  condition-2 ]  [ AFTER  { index-name-5 }  FROM
                               { data-name-7  }

{ index-name-6      }   BY   { data-name-9       }
{ data-name-8       }        { numeric-literal-6 }
{ numeric-literal-5 }

UNTIL  condition-3 ]
```

PERFORM is the means by which subroutines are executed in COBOL. The subroutines may be executed once, or a number of times, as determined by a variety of controls. A given paragraph may be PERFORMed by itself, in conjunction with another paragraph, control may pass through it in sequential operation, and it may be the object of a GO statement, all in the same program. The range of a PERFORM starts with the first executable statement of procedure-name-1 and continues in logical sequence through the last executable statement of:

a. THRU procedure-name-2, if specified, automatically sets up a return to the statement following the PERFORM statement.

b. Procedure-name-1 only, if procedure-name-2 is not specified automatically sets up a return to the statement following the PERFORM statement.

c. The automatic return is implied as immediately following the last statement in a PERFORM range.

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

Each data-name is a numeric elementary item described in the DATA DIVISION. All literals must represent numeric items with no positions to the right of the assumed decimal point.

There is no necessary relationship between procedure-name-1 and procedure-name-2 except that a consecutive sequence of operations is to be executed beginning at procedure-name-1 and ending with the execution of procedure-name-2. In particular, GO and PERFORM statements may only occur within procedure-name-1 and before the end of procedure-name-2. If there are two or more direct paths to the return point in procedure-name-1, then procedure-name-2 may be the name of a paragraph consisting solely of the EXIT statement, to which all of the procedure-name-1 paths must lead.

If the object program control passes to procedure-name-1 or procedure-name-2 from a statement other than a PERFORM, the procedure(s) will be accomplished and control will fall through to the next sentence following the procedure(s). If procedure-name-2 consists of an EXIT, program control will pass to the next sentence following procedure-name-2.

If a statement within procedure-name-1 or procedure-name-2 contains a nested PERFORM, object program control will pass to the procedure-name contained in the nested statement and the procedure will be accomplished. Program control will automatically return to the next sentence following the executed PERFORM statement. Nested PERFORM statements are allowed to any reasonable depth. Nesting depth is dictated by STACK size in the object program. Additional depth may be obtained by use of an MCP CORE card at object program execution time, or by the MEMORY SIZE clause, to give the object program additional memory to increase the STACK size. However, the procedure named must return to the statement following the previously executed PERFORM and cannot contain a GO out of range of procedure-name-1 or procedure-name-2.

A PERFORM statement appearing in an overlay section can have within its range only the following:

a. Procedures within the fixed (non-overlayable) segment of the program.

b. Procedures within the overlay segment containing the PERFORM.

A PERFORM statement appearing within the fixed segment of the program can range unrestricted to any point in the program.

Option 1 is the basic PERFORM statement. A procedure referred to by this type of PERFORM statement is executed once and then control passes to the statement following the PERFORM statement.

Option 2 is the TIMES option and, when used, the procedures are performed the number of times specified by data-name-1 or integer-1. Data-name-1 cannot be described as larger than 6 digits in length. The value of data-name-1 or integer-1 must be positive. Control is transferred to the statement following the PERFORM statement. If the value is zero, control passes immediately to the statement following the PERFORM sentence. Once the PERFORM statement has been initiated, any reference to or manipulation of data-name-1 will not affect the number of times the procedures are executed.

Option 3 is the UNTIL option. The specified procedures are performed until the condition specified by the UNTIL condition is TRUE. At this time, control is transferred to the statement following the PERFORM statement. If the condition is TRUE at the time that the PERFORM statement is encountered, the specified procedure is not executed.

Option 4 is the VARYING option. This option is used when it is desired to augment the value of one or more data-names or index-names in an orderly fashion during the execution of a PERFORM statement. When index-names are used, the FROM and BY clauses have the same effect as in a SET statement.

In option 4 where only one condition is required to control the number of iterations that a procedure is to be PERFORMed, the following actions take place:

a. Data-name-1 is set at the start of the PERFORM to a starting value as contained in data-name-2 (or numeric-literal-1).

b. Condition-1 is compared for an EQUAL condition. If condition-1 is true, control passes to next statement.

c. Procedure-name will be executed one time.

d. Data-name-3 is added to the contents of data-name-1.

e. Loop to step b above.

The above cycle continues until an equal comparison occurs, at which point program control directly passes to the next sentence following the executed PERFORM statement.

In option 4 where two conditions are required to control the number of iterations that a given procedure is to be PERFORMed, the following actions occur:

a. Data-name-1 and data-name-4 are set at the start of the PERFORM to starting values as contained in data-name-2 (or numeric-literal-1) and data-name-5 (or numeric-literal-3) respectively.

b. Condition-1 is compared to data-name-1 and:

   1) If an equal condition occurs, control is passed to the next sentence following the executed PERFORM statement, or else:

2) Condition-2 is compared to data-name-4 and:

   a) If an equal condition occurs, data-name-4 is set to the value contained in data-name-5. Data-name-3 is added to the data-name-1 and loop to step b above, or else:

   b) Procedure-name will be executed one time, after which data-name-6 is added to data-name-4 and loop to step a above.

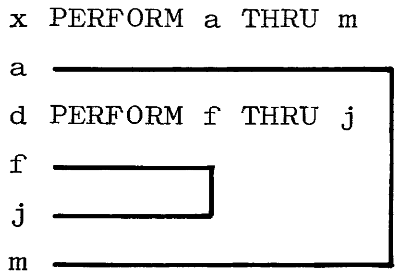
The above cycle continues until an equal comparison occurs, at which point program control directly passes to the next sentence following the executed PERFORM statement.

NOTE
Data-name-3 and data-name-6
cannot contain zeros.

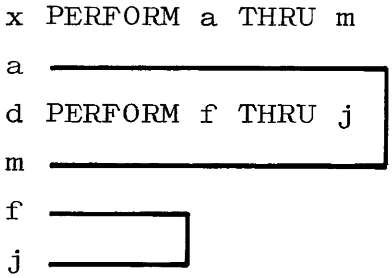
In option 4 where three conditions are required to control the number of iterations that a given procedure is to be PERFORMed, the mechanism is the same as for two-conditional control except that data-name-7 goes through a complete cycle each time that data-name-6 is added to data-name-4, which in turn goes through a complete cycle each time that data-name-1 is varied.

After the completion of Option 4, data-name-4 and data-name-7 contain their initial values, while data-name-1 contains a value which exceeds its last used setting by one increment or decrement unless condition-1 is TRUE when the PERFORM statement is entered, in which case data-name-1, data-name-4 and data-name-7 all contain their initial values.

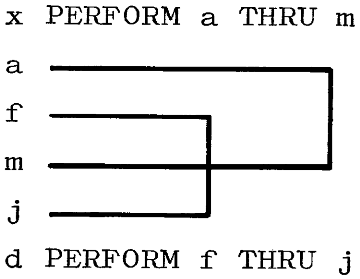
Since the return control information is placed in the stack rather than directed through instruction address modification, a PERFORM statement executed within the range of another PERFORM is not restricted in the range of paragraph names it may include. The examples shown on the following page are permitted and will execute correctly.



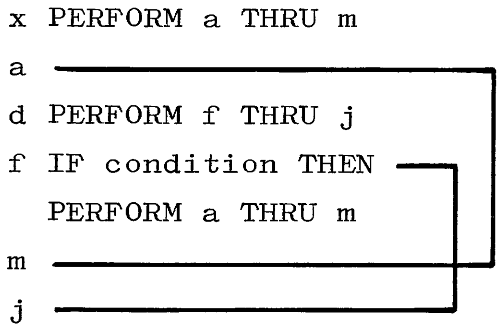
```
x PERFORM a THRU m
a
d PERFORM f THRU j
f
j
m
```



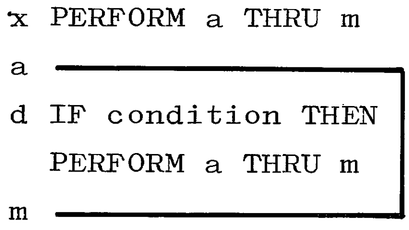
```
x PERFORM a THRU m
a
d PERFORM f THRU j
m
f
j
```



```
x PERFORM a THRU m
a
f
m
j
d PERFORM f THRU j
```



```
x PERFORM a THRU m
a
d PERFORM f THRU j
f IF condition THEN
  PERFORM a THRU m
m
j
```



```
x PERFORM a THRU m
a
d IF condition THEN
  PERFORM a THRU m
m
```

READ.

The function of this verb is twofold, namely:

a. When processing sequential input files, a READ statement will cause the next sequential record to be moved from the input buffer area to the actual work area, thus making the record available to the program if the file has been declared BLOCKED, or, if an ALTERNATE AREA has been ASSIGNed. An input buffer area is not used where records are unblocked or an ALTERNATE AREA has not been ASSIGNed, therefore, all sequential records will be physically read into the work area of the program. Physical READs are performed as a function of the MCP. The READ statement permits the performance of a specified imperative statement when an end-of-file condition is detected by the MCP.

b. For random file processing, the READ statement communicates with the MCP to explicitly cause the reading of a physical record from a disk file and also allows performance of a specified imperative statement if the contents of the associated ACTUAL KEY data item is found to be invalid.

The construct of this verb is:

```
READ file-name RECORD [ { WITH LOCK      } ] [ { AT END      }
                      [ { INTO data-name } ] [ { INVALID KEY }

   any imperative statement   [ { OTHERWISE }   any statement ] ]
                              [ { ELSE      }                 ] ]
```

The AT END of file clause is used only for non-disk files or for disk files being processed in the sequential access mode. NEXT-SENTENCE is implied if AT END is not explicitly stated.

If, during execution of a READ statement with AT END, the logical End-of-File is reached and an attempt is made to read that file, the statement specified in the AT END phrase is executed. After the execution of the imperative statement of the AT END phrase, a READ statement for that file must not be given without prior execution of a CLOSE statement and an OPEN statement for that file.

The WITH LOCK option makes the current record (physical record) unavailable to other programs and is only applicable for Shared Disk MCP usage.

The INTO and WITH LOCK options are mutually exclusive.

When the AT END clause is specified in a conditional sentence, the clause can be thought of as being preceded by an IF and that all exits within the sentence are controlled by using the rules pertaining to the matching of IF...ELSE pairs. For example:

```
IF AAA = BBB THEN READ FILE-A, AT END GO TO WRAP-UP,
   ELSE GO TO PROCESS-THE-RECORD, ELSE STOP RUN.
```

a. When AAA does not equal BBB, control will be passed to STOP RUN.

b. When AAA equals BBB, FILE-A is read, end-of-file is tested and if the result is "TRUE" program control will be transferred to the WRAP-UP procedure, however, a result of "FALSE" will cause program control to be transferred to PROCESS-THE-RECORD.

The INVALID KEY applies to files that are ASSIGNed to disk. The access of the file is controlled by the value contained in ACTUAL KEY.

An AT END or INVALID KEY clause <u>must</u> be specified when reading a file described as containing FILE-LIMITS.

The INTO option may only be used when the input file contains records of one type. The data-name must be the name of a WORKING-STORAGE area or output record area.

An OPEN statement must be executed for a file prior to the execution of the first READ statement for that file.

When a file consists of more than one type of logical record, these records automatically share the same storage area and are equivalent to an implicit redefinition of the area. Only the information that is present in the current record is available.

If the INTO option is specified, the current record is MOVEd from the input area to the area specified by data-name according to the rules for the MOVE statement without the CORRESPONDING option. If multiple 01 levels are declared in the file description, the size of the first 01 level is used.

When the INTO option is used, the record being read is available in both the data area associated with data-name and the input record area.

If a file described with the OPTIONAL clause is not present, the imperative statement in the AT END phrase is executed on the first READ. The standard End-of-File procedures are not performed. (See the OPEN and USE statements, and the FILE-CONTROL paragraph in the ENVIRONMENT DIVISION.)

If the end of a magnetic tape file is recognized during execution of a READ statement, the following operations are carried out:

a. The standard ending reel label procedure and the user's ending reel label procedure, if specified by the USE statement, are carried out. The order of execution of these two procedures is specified by the USE statement.

b. A tape swap is performed.

c. The standard beginning reel label procedure and the user's beginning label procedure, if specified, are executed. The order of execution is again specified by the USE statement.

d. The first data record on the new reel is made available.

READ with INVALID KEY is used for disk files in the random access mode. The READ statement implicitly performs the functions of the SEEK statement, except for the function of the KEY CONVERSION option for a specific disk file. If the contents of the associated

ACTUAL KEY data item is out of the range indicated by FILE LIMITS, the INVALID KEY phrase will be executed.

For random disk files, the sensing of an INVALID KEY does not preclude further READs on that file nor need it be closed and reopened before doing so.

If a READ parity error occurs, the MCP will pass the record back and forth under the read head until the record is successfully read, or until a specified amount of retry passes has been reached. If the parity error is unrecoverable, the MCP will branch to the USE... routine provided by the programmer. If a USE... routine is not found, the MCP will notify the systems operator to discontinue (DS) or dump the contents of memory pertaining to the program and then discontinue (DP) the run when the TERM option of the MCP (terminate automatically) is "OFF". If the TERM option is "ON" during the discovery of an unrecoverable magnetic tape parity error, a program which doesn't contain a USE... routine will be automatically terminated and all tapes re-wound, otherwise, the tapes will be setting one record beyond the parity when NO ALTERNATE AREAS are explicitly or implicitly specified for the file.

Object code will be generated, automatically at compile time, within every program declaring blocked logical records, which will cause de-blocking of physical records under control of the object program itself. An exception to this procedure of unblocking can be achieved by declaring "MCPB" in the dollar sign ($) card (see Section 8), however, speed will be sacrificed for a small savings in core used by the unblocking routine for the specific file.

RELEASE.

The function of this verb is to cause records to be transferred to the initial phase of a SORT operation.

The construct of this verb is:

```
RELEASE record-name [FROM data-name]
```

A RELEASE statement may only be used within the range of an input procedure associated with a SORT statement.

In the FROM option, the data-name must refer to a WORKING-STORAGE, or an input-record area.

Record-name and data-name must name different memory areas when specified.

The RELEASE statement causes the contents of record-name to be released to the initial phase of a sort. Record-name will be transferred to the specified sort-file (SD) and becomes controlled by the sort operation.

In the FROM option, the contents of data-name are MOVEd to record-name, then the contents of record-name are released to the initial phase of a sort. Moving takes place according to the rules specified for the MOVE statement without the CORRESPONDING option. The record-name area will not contain intelligible data after the MOVE, however, the information in data-name is still available.

After the RELEASE has been executed, record-name is no longer available. When control passes from the input procedure, the SD file consists of all those records that were placed in it by the execution of RELEASE statements.

RETURN.

The function of this verb is to obtain sorted records from the final phase of a SORT operation.

The construct of this verb is:

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

   [AT END  any statement]
```

File-name must be a sort file with a Sort File Description (SD) entry in the DATA DIVISION.

A RETURN statement may only be used within the range of an output procedure associated with a SORT statement for file-name.

The INTO option may only be used when the input file contains just one type of record. The data-name specified must be the name of a WORKING-STORAGE, or an output-record area.

Records automatically share the same area when a file consists of more than one type record and only the information pertinent to the current record is available.

The execution of the RETURN statement causes the next record, in the order specified by the Keys listed in the SORT statement, to be made available for processing in the record area associated with the SORT file (SD).

Moving is performed according to the rules specified for the MOVE statement without the CORRESPONDING option.

When the INTO option is specified, the sorted data is available in both the input-record area and the data-area specified by data-name.

RETURN statements may not be executed within the current SORT output procedure after the AT END clause has been executed.

SEARCH.

The function of this verb is to cause a search of a table to locate a table-element that satisfies a specific condition and, in turn, to adjust the associated index-name to indicate that table-element.

The construct of this verb has two options which are:

Option 1:

```
SEARCH data-name-1  [ VARYING  { index-name-1 } ]
                               { data-name-2  }

[ AT END imperative statement-1 ]

  WHEN condition-1  { imperative statement-2 }
                    { NEXT SENTENCE          }

[ WHEN condition-2  { imperative statement-3 }   ... ]
                    { NEXT SENTENCE          }
```

Option 2:

```
SEARCH ALL data-name-3  [AT END imperative statement-4]

  WHEN condition-3  { imperative statement-5 }
                    { NEXT SENTENCE          }
```

Data-name-1 and data-name-3 may not be subscripted or indexed, but their descriptions must contain an OCCURS clause and an INDEXED BY option.

When Option 2 is specified, the description of data-name-3 may optionally contain the ASCENDING/DESCENDING KEY clause.

When using the VARYING option, data-name-2 must be described as USAGE IS INDEX, or as the name of a numeric elementary item described without any positions to the right of the assumed decimal point. Data-name-2 will be incremented at the same time as the occurrence number (and by the same amount) represented by the index-name associated with data-name-1.

When using Option 1, condition-1, condition-2, etc., may be comprised of any conditional as described by the IF verb.

When using Option 2, condition-3 may consist of a relational condition incorporating the relation EQUAL, or a condition-name condition where the VALUE clause that describes the condition-name contains only a single literal. Condition-3 may be a compound condition formed from simple conditions of the type just mentioned, with AND being the only acceptable connective.

When using Option 2, any data-name that appears in the KEY option of data-name-3 may appear as the subject <u>or</u> object of a test, <u>or</u> be the name of the conditional variable with which the tested condition-name is associated.

When using Option 1, a serial type search operation takes place, starting with the current index setting. The search is immediately terminated if, at the start of execution of the statement, the index-name associated with data-name-1 contains a value that corresponds to an occurrence number that is greater than the highest permissable occurrence number for data-name-1. Then, if the AT END option is specified, imperative statement-1 is executed; if AT END is not specified, control passes to the NEXT SENTENCE.

When using Option 1, if at the start of execution of the SEARCH statement, the index-name associated with data-name-1 contains a value that corresponds to an occurrence number that is <u>not</u> greater than the highest permissible occurrence number for data-name-1, the

SEARCH statement will begin evaluating the conditions in the order that they are written, making use of index settings wherever specified, to determine the occurrences of those items to be tested. If none of the conditions are satisfied, the index-name for data-name-1 is incremented to obtain a reference to the next occurrence. The process is repeated using the new index-name setting for data-name-1 which corresponds to a table element which exceeds the last setting by one more occurrence until such time as the highest permissible occurrence number is exceeded, in which case the SEARCH terminates as indicated in the previous paragraph.

When using Option 1, if one of the conditions is satisfied upon its evaluation, the SEARCH terminates immediately and the imperative statement associated with that condition is executed; the index-name remains set at the occurrence which caused the condition to be satisfied.

In Option 1 and 2, if the specified imperative statements do not terminate with a GO statement then program control will pass to the next sentence after the execution of the imperative statement.

In the VARYING option, if index-name-1 appears in the INDEXED BY option of data-name-1, then that index-name will be used for the SEARCH, otherwise, the first index-name given in the INDEXED BY option of data-name-1 will be used. If index-name-1 appears in the INDEXED BY option of another table entry, the occurrence number represented by index-name-1 is incremented by the same amount as, and at the same time as, the occurrence number represented by the index-name associated with data-name-1 is incremented.

In Option 2, the initial setting of the index-name for data-name-3 is ignored, the effect being the same as if it were SET to 1.

In Options 1 and 2, if data-name-1 and data-name-3 is an item in a group, or a hierarchy of groups, whose description contains an OCCURS clause, then each of these groups must also have an index-name associated with it. The settings of these index-names are used throughout the execution of the SEARCH statement to refer to data-names-1 and 3, or items within its structure. These index settings are not modified by the execution of the SEARCH statement (unless stated as index-name-1) and only the index-name associated with data-names-1 and 3 (and data-name-2 or index-name-1) is incremented by the SEARCH. Figure 5-9 provides an example of SEARCH operation as related to Option 1.

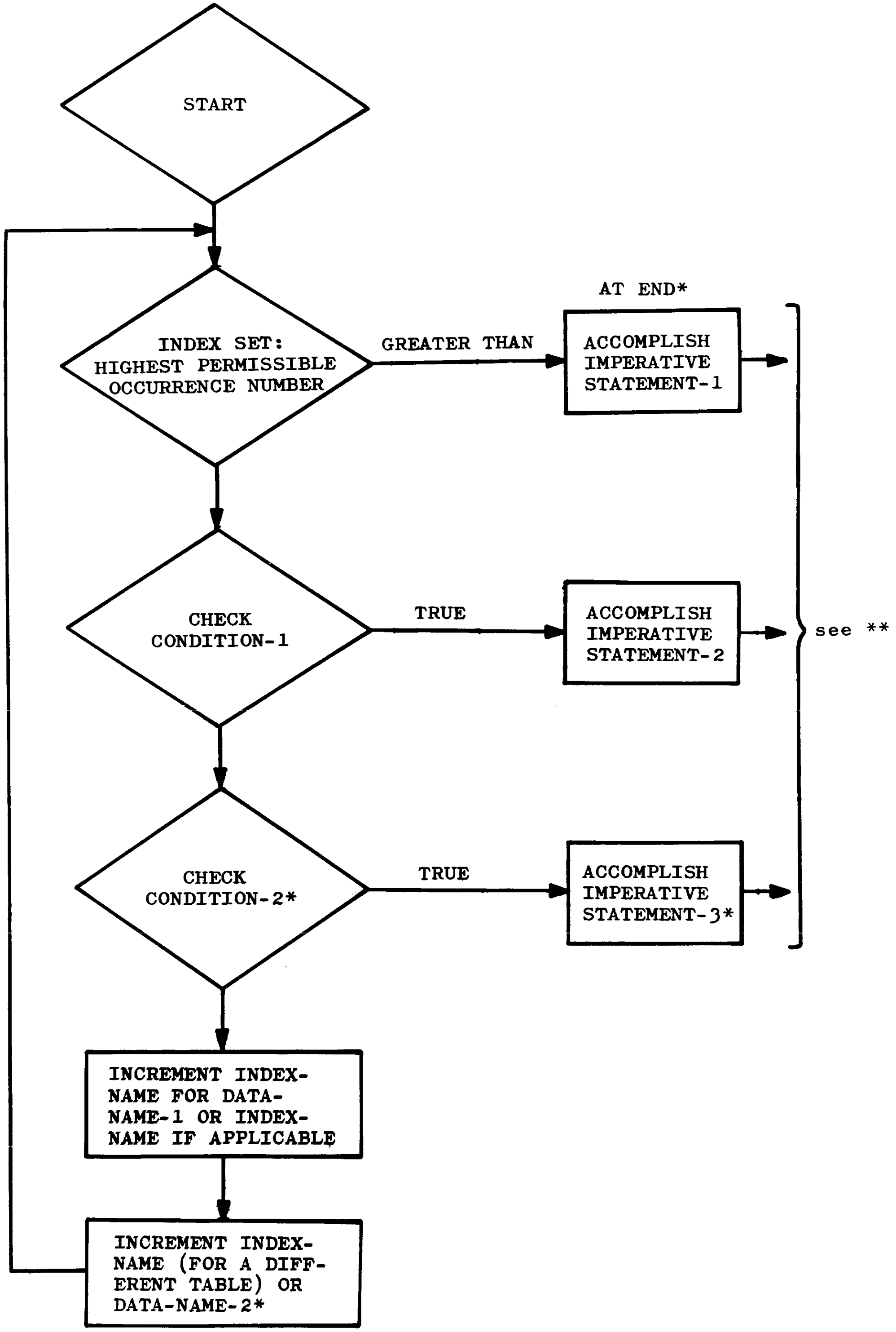


Figure 5-9. Example of SEARCH Operation Relating to Option 1

* These operations are only included when called for in the SEARCH statement.

** Each of the control transfers is to NEXT SENTENCE unless the imperative statement ends with a GO statement.

SEEK.

The function of this verb is to initiate the accessing of a disk file record for subsequent reading and/or writing.

The construct of this verb is:

```
SEEK file-name RECORD [WITH KEY CONVERSION] [LOCK]
```

The specification of the KEY CONVERSION clause indicates that the user provided USE FOR KEY CONVERSION section in the DECLARATIVE SECTION is to be executed prior to the execution of the SEEK statement. If there are no DECLARATIVES for KEY CONVERSION in a SEEK statement, then the KEY CONVERSION clause will be ignored.

A SEEK statement pertains only to disk storage files in the random access mode and may be executed prior to the execution of each READ and WRITE statement. Use of the LOCK clause makes the record unavailable to other programs (shared disk MCP only).

The SEEK statement uses the contents of the data-name in the ACTUAL KEY clause for the location of the record to be accessed. At the time of execution, the determination is made as to the validity of the contents of the ACTUAL KEY data item for the particular disk storage file. If the key is invalid, the imperative statement in the INVALID KEY clause of the next executed READ or WRITE statement for the associated file is executed.

Two SEEK statements for a disk storage file may logically follow each other. Any validity check associated with the first SEEK statement is negated by the execution of a second implicit or implied SEEK statement.

An implied SEEK is executed by the MCP whenever an explicit SEEK is missing for the specified record. An implied SEEK never executes any USE KEY CONVERSION Declaratives.

If a READ/WRITE statement for a file ASSIGNed to DISK is executed, but an explicit SEEK has not been executed since the last previous READ or WRITE for the file, then the implied SEEK statement is executed as the first step of the READ/WRITE statement.

An explicit alteration of ACTUAL KEY after the execution of an explicit SEEK has been performed, but prior to a READ/WRITE, will cause the initiation of an implied SEEK of the initial record in the sequence. For example,

a. If ACTUAL KEY is 10, then
b. READ record 10, then
c. MOVE 50 to ACTUAL KEY, then
d. WRITE record 50.

An implied SEEK of record 50 will be performed between actions c. and d. above.

SET.

The function of this verb is to establish reference points for table handing operations by setting index-name values associated with table elements.

The construct of this verb has two options which are:

Option 1:

```
SET     { index-name-1 }  [{ index-name-2 }   ... ]
        { data-name-1  }  [{ data-name-2  }       ]

    TO    { index-name-3 }
          { data-name-3  }
          { literal-1    }
```

Option 2:

```
SET   index-name-4  [index-name-5 ...]

    { UP BY   }   { data-name-4 }
    { DOWN BY }   { literal-2   }
```

All data-items must be either index-data-names or numeric elementary items described without any positions to the right of the assumed decimal point, except that data-name-4 must not be an index-data-name. When a literal is used, it must be a positive integer. Index-names are considered related to a given table and are defined by being specified in the INDEXED BY clause.

An index-data-name cannot be SET...TO... a literal or to a data-name.

A data-name cannot be SET...TO... an index-data-name, a literal or another data-name. A data-name can only be SET to an index-name.

Literals cannot be SET...TO anything.

The SET verb appears somewhat similar to the MOVE but has a major difference in that the receiving field appears as the first operand(s) in the statement. For example:

```
SET A TO B
```

The above statement causes the contents of A to change to the value contained in B. Series statements may result in more efficient object code than separate statements. For example:

```
SET A, C, D, E, F TO B
```

Depending on the operands in a SET statement, code generated will vary from a single MVN through a series of MVN, MUL and DIV instructions. Because of this, care must be used in determining what type of receiving operand is going to be SET to what type of sending operand, since this is the primary step in calculating the location within the row. For example:

```
SET INDEX-DATA-NAME-A TO INDEX-A
SET INDEX-B TO INDEX-DATA-NAME-A
```

Both of the above statements are, by COBOL definition, plain MOVEs and unless the two indexes refer to rows of exactly the same size, will probably not result in an address which the programmer has perceived. If instead, the statement had been written: SET INDEX-B TO INDEX-A, the necessary MOVE, DIVIDE and MULTIPLY instructions would be generated to reduce the "sending" index to a relative occurrence (subscript) and then to expand it to the receiving address.

SORT.

The function of this verb is to sort a magnetic tape or disk input file of records by transferring such data into a disk sort-file (work file) and sorting those records on a set of specified keys. The final phase of the sort operation makes each record available from the sort-file, in sorted order, to an output procedure or to a magnetic tape or disk output file.

The construct of this verb is:

```
SORT  file-name-1

[ { PURGE }
  { RUN   }        ON ERROR ]
  { END   }

   ON { DESCENDING } KEY data-name-1 [data-name-2...]
      { ASCENDING  }

[  ON { DESCENDING } KEY data-name-3 [data-name-4...] ]
      { ASCENDING  }

{ INPUT PROCEDURE IS section-name-1 [ { THRU    } section-name-2] }
{                                     { THROUGH }                 }
{ USING file-name-2 [ { LOCK    } ]                               }
{                     { PURGE   }                                 }
{                     { RELEASE }                                 }

{ OUTPUT PROCEDURE IS section-name-3 [ { THRU    } section-name-4] }
{                                      { THROUGH }                 }
{ GIVING file-name-3 [ { LOCK    } ]                               }
{                      { RELEASE }                                 }
```

File-name-1 must be described in a Sort File Description (SD) entry in the DATA DIVISION and file-names-2 and 3 must be described in a File Description (FD) entry.

Section-name-1 specifies the name of the input procedure to be used before passing each record to the sort-file, while section-name-3 specifies the output procedure to be used to obtain each sorted record from the sort-file.

Each data-name must represent data-items described in records associated with file-name-1. Data-names following the word KEY are listed from left to right in the order of decreasing significance without regard as to their division into optional KEY clauses.

A maximum of 40 KEYs are allowed in each SORT statement and each KEY may be comprised of up to 99 digits or characters in length, except for signed KEYs, which may be comprised of up to 50 characters. The total accumulated size of KEYs cannot exceed 3,960. characters or digits.

The PROCEDURE DIVISION of a source program may contain more than one SORT statement appearing anywhere in the program, except in the DECLARATIVES portion or in the input/output procedures associated with a sort statement.

The input procedure must consist of one or more sections that are written consecutively and which do not form a part of an output procedure. The input procedure must include at least one RELEASE statement in order to transfer records to the sort-file after the object program has accomplished the required input data manipulation specified in the procedure. Input procedures can select, create and/or modify records, one at a time, as specified by the programmer.

There are three restrictions placed on procedural statements within an input or output procedure:

a. The procedure must not contain any SORT statements.

b. The input or output procedures must not contain any transfers of program control outside the range of the procedure; ALTER, GO and PERFORM statements within the procedure are not permitted to refer to procedure-names outside of the input or output procedure.

c. The remainder of the PROCEDURE DIVISION must not contain any transfers of program control to points within the input or output procedure; ALTER, GO, and PERFORM statements in the remainder of the PROCEDURE DIVISION must not refer to procedure-names within the range of the input or output procedure.

The output procedure must consist of one or more sections that are written consecutively and which do not form a part of an input procedure. The output procedure must include at least one RETURN statement in order to make each sorted record available for processing after the file has been sorted and the object program has accomplished the required output data manipulation specified in the procedure. Output procedures can select, create and/or modify records, one at a time, as they are being returned from the sort-file.

When the ASCENDING clause is specified, the sorted sequence of the affected records is from the lowest to the highest value according to the collating sequence of the B 2500/B 3500, per specified KEY.

When the DESCENDING clause is specified, the sorted sequence of the affected records is from the highest to the lowest value according to the collating sequence of the B 2500/B 3500, per specified KEY.

The SD record description of the sort-file must contain fully defined data-name KEY items in the relative positions of the record as applicable. A rule to follow when using these KEY items is that when a KEY item appears in more than one type of record, the data-names must be relatively equivalent in each record and may not contain, or be subordinate to, entries containing an OCCURS clause.

When an input procedure is specified, object program control will be passed to that procedure automatically as an implicit function of encountering the generated SORT verb object code compiled into the program. The compiler will insert a return-to-the-sort mechanism at the end of the last section in the input procedure and when program control passes the last statement of the input procedure, the records that have been RELEASED to file-name-1 commence being sorted.

If the USING option is specified, all records residing in file-name-2 will be automatically transferred to file-name-1 upon encountering the generated SORT verb object code. At the time of execution of the SORT statement, file-name-2 must not be OPEN. The SORT statement automatically performs the function necessary to OPEN, READ, USE and CLOSE file-name-2. If file-name-2 is a disk file, it must be in the Disk Directory before the SORT Intrinsic is called.

When an output procedure is specified, object program control will be passed to that procedure automatically as an implicit function when all records have become sorted. The compiler will insert a return-to-the-object program mechanism at the end of the last section in the output procedure and when program control passes the last statement of the output procedure, the object program will execute the next statement following the pertinent SORT statement.

If the GIVING option is specified, all sorted records residing in file-name-1 are automatically transferred to the OUTPUT file as specified in file-name-3. At the time of execution of the sort statement, file-name-3 must not be OPEN. File-name-3 will be automatically OPENed before the sorted records are transferred from the sort-file and in turn, will be automatically CLOSEd after the last record in the sort-file has been transferred.

The ON ERROR option is provided to allow programmers some control over irrecoverable parity errors when INPUT/OUTPUT PROCEDURES are not present in a program. PURGE will cause all records in a block containing an irrecoverable parity error to be dropped and processing will be continued after a SPO message giving the relative position in the file of the bad block has been printed. This option is always assumed if no other has been defined. RUN will cause the bad block to be used by the program and will provide the same SPO message as defined for PURGE. END will cause the usual DS or DP SPO message.

The PURGE, LOCK, and RELEASE options may be used to specify the type of file close on file-name-2 and file-name-3 (see CLOSE, page 5-33). The options only apply to the USING/GIVING options.

Example:

```
SORT file-name-1 ASCENDING KEY data-name-1
USING file-name-3 PURGE
GIVING file-name-3 LOCK.
```

Beginning and ending label USE procedures are provided as follows when INPUT/OUTPUT PROCEDURES are present in the SORT statement:

a. OPEN INPUT file-name.
   USE...(The programmer's USE procedure will be invoked.)

b. OPEN OUTPUT file-name.
   USE...(The programmer's USE procedure will be invoked.)

c. CLOSE input-file-name.
   USE...(The programmer's USE procedure will be invoked, however, the contents of the ending input label will not be available to the USE procedure.)

d. CLOSE output-file-name.
   USE...(The programmer's USE procedure will be invoked; however, the ending label will have been written prior to executing the USE procedure.)

NOTE

The above actions provide USE on label facilities at beginning and ending of files but not when switching reels of multi-reel files.

STOP.

The function of this verb is to halt the object program temporarily or to terminate execution.

The construct of this verb is:

```
STOP   { RUN     }
       { literal }
```

If the word RUN is used, then all files which remain OPEN will be CLOSED automatically. Files ASSIGNED to DISK will be CLOSED WITH PURGE and all others will be CLOSED WITH RELEASE. All storage areas for the object program are returned to the MCP and the job is then removed from the MCP Mix.

The STOP RUN is not used for temporary stops within a program. STOP RUN must be the last statement of the program execution sequence.

If the literal option is used, the literal will be DISPLAYed on the message printer and the program will be suspended. When the operator enters the MCP continuation message mix-index AX, program execution resumes with the next sequential operation. This option is normally used for operational halts to cause the system's operator to physically accomplish an external action.

If a STOP statement with the RUN option appears in an imperative statement, then it must appear as the only statement or last statement in the imperative statement.

SUBTRACT.

The function of this verb is to subtract one, or the sum of two or more, numeric data items from another item, and set the value of an item equal to the result(s).

The construct of this verb has three options which are:

Option 1:

```
SUBTRACT    { literal-1   }   [ { literal-2   }  ... ]  FROM
            { data-name-1 }   [ { data-name-2 }      ]

  data-name-m [ROUNDED]  [ data-name-n [ROUNDED] ... ]

[ ON SIZE ERROR any statement   [ { OTHERWISE }   statement ] ]
[                               [ { ELSE      }             ] ]
```

Option 2:

```
SUBTRACT    { literal-1   }   [ { literal-2   }    ... ]  FROM
            { data-name-1 }   [ { data-name-2 }        ]

{ literal-m   }     GIVING   data-name-n [ROUNDED]
{ data-name-m }

[ ON SIZE ERROR any statement  [ { OTHERWISE }   statement ] ]
[                              [ { ELSE      }             ] ]
```

Option 3:

```
SUBTRACT  { CORR          }  data-name-1 FROM data-name-2
          { CORRESPONDING }

[ROUNDED]  [ ON SIZE ERROR any statement  [ { OTHERWISE }
                                            { ELSE      }

statement ] ]
```

In Options 1 and 2, the data-names used must refer only to elementary numeric items. If Option 2 is used, the data-description of data-name-n may contain editing symbols, except when data-name-n also appears to the left of GIVING.

All rules specified under the ADD statement with respect to the operand size, presence of editing symbols in operands, the ON SIZE ERROR option, the ROUNDED option, the GIVING option, truncation, the editing of results, the handling of intermediate results, and the CORR or CORRESPONDING option apply to the SUBTRACT statement.

When the GIVING option is not used, a literal may not be specified as the minuend.

When dealing with multiple subtrahends, the effect of the subtraction will be as if the subtrahends were first summed, and then the sum subtracted from the minuends.

TRACE.

The function of this verb is to create documentation of all normal and/or control mode processing events and to output this data on a line printer.

The construct of this verb is:

$$\underline{\text{TRACE}} \quad \left\{ \begin{matrix} \underline{0} \\ \underline{1} \\ \underline{2} \\ \underline{3} \\ \underline{20} \end{matrix} \right\}$$

When a TRACE statement is encountered during object program execution, one of the following actions will take place at that point in the program:

a. The 0 option will turn the TRACE off.

b. The 1 option will cause a TRACE of all normal mode instructions until such time as any of the other options are encountered.

c. The 2 option will cause a TRACE of all control mode instructions until such time as any of the other options are encountered.

d. The 3 option will cause a TRACE of all control and normal mode instructions until such time as any of the other options are encountered.

e. The 20 option will cause a memory dump to be taken of locations base relative to the program's memory assignment. Processing will continue after the memory "snap-shot."

If TRACE is turned ON during multiprocessing, all processing will be traced according to the selected option. Output will go to a printer regardless of its MCP job assignment to a specific program in the Mix.

The following is an example of TRACE output:

```
1   2       3      4       5        6     7         8  9  10  11  12   13
N   15036   GEQ            16900                                  <
N   15044   MVN    0705    10004    1S    12010     2S            <   0    +12345
N   15062   OFL            16024                                  <
N   16024   BCT    0354                                           <
    10000   MVA    0202    25       AL    10000     A             >        25
```

Column Explanation

1 Normal State = N, Control State = blank.

2 Address of Instruction in Memory.

3 Operation Code. The number of an invalid Operation Code.

4 AF-BF Field.

5 A-Address.

6 AI-AC Field:

AI-1,2,3 = Index Register.
AC-S = Signed Numeric Data.
A = Alphanumeric Data.
I = Indirect Address.
Blank = Unsigned Numeric Data.

For literal types of data in the A-Address Field, the following descriptions appear in the combined AI-AC Fields:

AL = Alphanumeric Literal.
NL = Numeric Literal.
SL = Signed Literal.
FL = Floating Point Literal.

7 B-Address.

8 BI-BC Field:

BI-1,2,3 = Index Register.
BC-S = Signed Numeric Data.
A = Alphanumeric Data.
I = Indirect Address.
Blank = Unsigned Numeric Data.

9 C-Address.

10 CI-CC Field:

CI-1,2,3 = Index Register.
CC-S = Signed Numeric Data.
A = Alphanumeric Data.
I = Indirect Address.
Blank = Unsigned Numeric Data.

11 Status of Compare Toggle.

12 Status of Overflow Toggle:

Blank = No Overflow Condition.
0 = Overflow Condition.

13 Contents of Result Field in memory to a maximum of 68 characters.

The Result Field in memory is defined for the various operators below:

OP	MNE	FIELD	OP	MNE	FIELD	OP	MNE	FIELD	OP	MNE	FIELD
01	INC	B	13	MVC	B	42	AND	C	91	SRD	IX1 (MCP)
02	ADD	C	14	MVR	B	43	ORR	C	92	RAD	A-SHOW 6
03	DEC	B	15	TRN	C	44	NOT	C	94	IIO	A-SHOW 18
04	SUB	C	16	SDE	B	45	CPA	B	95	RDT	A-SHOW 6
05	MPY	C	17	SDU	B	46	CPN	B	96	RCT	A-SHOW 6
06	DIV	C	18	SZE	B	49	EDT	C	97	STT	A-SHOW 6
09	MVL	C	19	SZU	B	80	FAD	C			
10	MVA	B	31	NTR	STK [IX3]	81	FSU	C			
11	MVN	B	40	BZT	A	82	FMP	C			
12	MVW	B	41	BOT	A	83	FDV	C			

Those not listed have no Result Field.

Numbers higher than nine in a 4-bit configuration are called UNDIGITS and will be represented on the TRACE and memory dump as:

Binary Number	Printed Character
10	A
11	B
12	C
13	D
14	E
15	F

The Memory Dump routine for TRACE 20 prints a starting location of the first of 100 digits/letters per line for as many lines as will encompass the entire program while resident in memory, plus its working storage and the area used for segmentation. The output data will be represented as per the EBCDIC Code Reference Table in appendix C of this manual. The following example of a memory dump reflects the technique utilized:

```
0900  E2C5C540C8  D6E640C9E3  40E6D6D9D2  E26F401234  56789-----  ----------
       S E E   H   O W   I T     W O R K   S ?   1234  56789
1000  F0F1F2F3F4  F5F6F7F8F9  FAFBFCFDFE  FF--------  ----------  ----------
       0 1 2 3 4   5 6 7 8 9  1011121314  15
2100  D0D1D2D3D4  D5D6D7D8D9  ----------  ----------  ----------  ----------
       0 1 2 3 4   5 6 7 8 9
2200  F1F2D0F9F8  D9--------  ----------  ----------  ----------  ----------
       1 2 0 9 8   9
```

UNLOCK.

The function of this verb is to make a shared disk file record available to other programs.

The construct of this verb is:

```
UNLOCK record-name
```

This statement does not WRITE a record; it only makes the record available.

USE.

The function of this verb is to specify procedures for any input/output error and/or label handling which are in addition to the standard procedures supplied by the MCP, to calculate the ACTUAL KEY for files assigned to DISK, and to accomplish various user required actions when a 12 punch (overflow) in the printer carriage control tape is encountered, and for Shared Disk stalemate conditions.

The construct of this verb has four options which are:

Option 1:

$$\underline{\text{USE}}\ \text{AFTER STANDARD}\ \underline{\text{ERROR}}\ \underline{\text{PROCEDURE}}\ \text{ON}\ \left\{\begin{array}{l}\text{file-name...}\\ \underline{\text{INPUT}}\\ \underline{\text{OUTPUT}}\\ \underline{\text{INPUT-OUTPUT}}\\ \underline{\text{I-O}}\\ \underline{\text{O-I}}\end{array}\right\}$$

Option 2:

$$\underline{\text{USE}}\quad \left\{\begin{array}{l}\underline{\text{AFTER}}\\ \underline{\text{BEFORE}}\end{array}\right\}\quad \text{STANDARD}\quad \left\{\begin{array}{l}\underline{\text{ENDING}}\\ \underline{\text{BEGINNING}}\end{array}\right\}$$

$$\left[\left\{\begin{array}{l}\underline{\text{REEL}}\\ \underline{\text{FILE}}\end{array}\right\}\right]\quad \underline{\text{LABEL}}\ \underline{\text{PROCEDURE}}\ \text{ON}\quad \left\{\begin{array}{l}\text{file-name...}\\ \underline{\text{INPUT}}\\ \underline{\text{OUTPUT}}\end{array}\right\}.$$

Option 3:

$$\underline{\text{USE}}\quad \left\{\begin{array}{l}\text{AT}\ \underline{\text{END}}\ \underline{\text{OF}}\ \underline{\text{PAGE}}\\ \text{FOR KEY}\ \underline{\text{CONVERSION}}\end{array}\right\}\quad \text{ON file-name-1}\ [\text{file-name-2...}].$$

Option 4:

USE ON STALEMATE ON file-name-3.

A USE statement, when present, must immediately follow a section header in the DECLARATIVE portion of the PROCEDURE DIVISION and must be followed by a period followed by a space. The remainder of the section must consist of one or more procedural paragraphs that define the procedures to be used.

If the file-name option is used as part of Option 2, the File Description entry for the file-name must not specify a LABEL RECORDS ARE OMITTED clause.

A USE statement specified for input and/or output files associated with the SORT verb will not be affected when executing the SORT unless an INPUT and/or OUTPUT PROCEDURE has been included in the program.

The USE statement itself is never executed rather, it defines the conditions calling for the execution of the USE procedures.

If neither REEL nor FILE is included in Option 2, the designated procedures are executed for both REEL and FILE labels. The REEL option is not applicable to mass storage files.

Within a given format, a file-name must not be referred to implicitly or explicitly in more than one USE statement.

USE procedures will be executed by the MCP:

a. After completing the standard I/O error retry routine (this applies only to option 1) the record in error has been read, thus another READ cannot appear in the USE section since the MCP is performing the section because of a previous READ which has been completed. Upon completion of the USE procedure, control is returned to the statement following the READ which detected the error condition. In the case of blocked or unblocked magnetic tape input, the tape will be sitting ready to read the next record as soon as the Option 1 procedure is completed. For example, if the user wishes to print the records which cannot be read because of an unrecoverable parity error, a procedure may be included as illustrated on page G-1, appendix G.

b. The USE AFTER STANDARD BEGINNING clause designates that the procedure following the clause must be called upon to check data on input magnetic tape beginning-file-labels, or to insert data as an output magnetic tape beginning-file-label before it is written. For example, if the user wishes to change fields within the BURROUGHS STANDARD LABEL and wants to add new fields to the Label Record, he may do so by including a procedure similar to that illustrated on page G-2, appendix G.

c. When the USE BEFORE STANDARD ENDING clause designates that a following procedure must be called upon to check user created data contained on input magnetic tape ending file labels or to insert data onto the user's portion of an output magnetic tape ending file label before it is written.

NOTE

USE AFTER STANDARD ENDING and
USE BEFORE STANDARD BEGINNING
are both illegal entries in
B 2500/B 3500 COBOL.

d. After a physical 12 punch is sensed on the printer's carriage control tape for the USE AT END OF PAGE statement.

e. Prior to any SEEK WITH KEY CONVERSION statement on files named in the USE FOR KEY CONVERSION statement.

References to common label items need not be qualified by a file-name within a USE statement. A common label item is defined as being an elementary data item that appears in every magnetic tape beginning and/or ending file-label record, but does not appear in any data record of the program.

A common label item must have the same name, description, and relative position in every magnetic tape file-label record and may only be referenced while in a USE...LABEL PROCEDURE for that file.

If the INPUT or OUTPUT option is specified, the USE...LABEL PROCEDUREs do not apply when files are described as having LABEL RECORDS OMITTED.

There must not be any reference to non-declarative procedures within a USE procedure. Conversely, in the non-declarative portion there must be no reference to procedure-names that appear in the declarative portion, except that a PERFORM statement may refer to a USE declarative, or to the procedures associated with such USE declaratives.

Option 2 is not applicable to disk files.

The USE AT END OF PAGE procedure in option 3 allows the object program to automatically branch to a central user routine at the time that a paper overflow condition on a line printer is sensed by the physical reading of a 12 punch in the carriage control tape. If two line printers are being used in the object program, both may use the same procedure where the actions to be fulfilled are identical.

Option 4 applies only to Shared Disk.

When shared disk is used, and two processors are accessing the same file, one processor may try to read a record which was locked by the second processor. At the same time, the second processor is trying to read a record locked by the first processor. This condition causes both processors to wait indefinitely, unless the USE ON STALEMATE option is used. This option allows the user to specify the action to be taken at that precise moment.

WAIT.

The function of this verb is to cause the suspension of an executing object program for a specified number of seconds.

The construct of this verb is:

$$\underline{\text{WAIT}} \quad \begin{Bmatrix} \text{literal} \\ \text{data-name} \end{Bmatrix}$$

WAIT may be executed when using any version of the MCP.

A WAIT statement specifying a literal will cause the executing object program to be suspended for that number of seconds and automatically become re-instated, after the specified period of time has expired, by the MCP.

The WAIT statement is particularily effective in continuous polling loops where polling is required every few seconds, thus releasing the intervening time to the other programs in the mix.

If data-name is specified as containing the WAIT value, it must be PICTUREd as PC 9(5) COMPUTATIONAL.

Reference should be made to the WAIT verb in section 6 (Data Communications) of this manual. That construct when specified, requires the presence of a version of the Data Communications MCP at object program execution time.

WRITE.

The function of this verb is to release a logical record for an output file. It is also used to vertically position forms in the printer. For mass storage files, the WRITE statement also allows the performance of a specified imperative statement if the contents of the associated ACTUAL KEY item are found to be invalid.

The construct of this verb has two options which are:

Option 1:

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

[                              { {integer-1  }      LINES          } ]
[  {AFTER }                    { {data-name-2}                     } ]
[  {BEFORE}    ADVANCING       {                                   } ]
[                              { [TO CHANNEL  {integer-2  }]       } ]
                               {              {data-name-3}        }
    [ TO   {ERROR    } ]
    [      {AUXILIARY} ]
```

Option 2:

```
WRITE   record-name  [{WITH LOCK     }]
                     [{FROM data-name}]

  [INVALID KEY any statement   [{OTHERWISE}    statement] ]
                               [{ELSE     }             ] ]
```

An OPEN statement for a file must be executed prior to executing the first WRITE statement for that file.

The record-name must be defined in the DATA DIVISION by means of a 01 level entry under the FD entry for the file. The record-name and data-name-1 must not be the same name, or be in two files that have the same record area.

The ADVANCING option allows the control of vertical positioning of each record on the printed page. The options are:

a. When LINES is used, data-name-2 must be declared as PC 99 COMPUTATIONAL or integer-1 must be a positive integral value of 00 THRU 99.

b. WRITE BEFORE ADVANCING is more efficient than AFTER ADVANCING.

c. When CHANNEL is used, data-name-3 or integer-2 must possess a positive integral value of 01 ... 11. Data-name-3 must be declared as PC 99 COMPUTATIONAL. The MCP will advance the line printers carriage to the carriage control channel specified.

Option 2 must be used for writing on disk files.

If the FROM option is specified, the data is moved from the area specified by data-name-1 in Option 1, and data-name in Option 2, to the output area, according to the rules specified for the MOVE statement without the CORR or CORRESPONDING option. After execution of the WRITE statement is completed, the information in the data-name following the word FROM is available, even though that record-name is not available.

When the WRITE statement is executed at object time, the logical record is released for output and is no longer available for referencing by the object program. Instead, the record area is ready to receive items for the next record to be written. If blocking is called for by the COBOL program, the records will be blocked by object code automatically generated at compile time. When the blocking area becomes full, or partially full at EOJ or EOF, the object program will transfer control to the MCP to cause the block to be physically written. An exception to this procedure of blocking can be achieved by declaring "MCPB" in the dollar sign ($) card (see section 8), however, speed will be sacrificed for a small savings in core used by the blocking routine for the specified file. Short blocks of records which were written during EOF or EOJ will be of no programmatic concern to the user when using the file as INPUT at a later period of time.

If a write error is detected during a magnetic tape write operation, the tape record in error will be erased and a rewrite will be attempted further down the tape until the record is finally written correctly. A punch or printer write error will result in a message to the operator. The COBOL programmer need not include any USE procedures to handle write errors.

The shortest allowable blocks which can be written on 7 and 9 channel magnetic tape units are 8 and 18 bytes respectively.

If a CLOSE statement has been executed for a file, any attempt to WRITE on the file until it is OPENed again will result in an error termination.

For files which are being accessed in a SEQUENTIAL manner, the INVALID KEY clause is executed when the end of the last segment of the file (last record) has been reached and another attempt is made to WRITE into the file. The last segment of a file is specified in the FILE-LIMITS clause or the FILE CONTAINS clause. Similarly, for files being accessed in a RANDOM manner, the INVALID KEY clause will be executed whenever the value of the ACTUAL KEY is outside the defined limits. An INVALID KEY entry must be specified when writing to a file described as containing FILE-LIMITS.

Records will be written onto DISK in either a SEQUENTIAL or RANDOM manner according to the rules given under ACCESS MODE. For RANDOM accessing, SEEK statements will be explicitly used for record determination as defined under ACCESS MODE, SEEK, and READ.

If the size and blocking of records being accessed in a RANDOM manner is such that a WRITE statement must place a record into the middle of a block without disturbing the other contents of the block, then an implicit SEEK will be given to load the block desired (if an explicit SEEK has not been given). If the file is being processed for INPUT/OUTPUT, then either an explicit or implicit SEEK for a READ statement will suffice to load the block between the READ and WRITE statements.

If the value of the ACTUAL KEY is changed after a SEEK statement has been given and prior to the WRITE statement, an implied SEEK will be performed and the WRITE will use the record area selected by the implied SEEK as the output record area. The value contained in ACTUAL KEY will not be affected.

For RANDOM access, when records are unblocked, the use of a SEEK statement related exclusively to WRITE is unnecessary, and may result in an extra loading of the record from disk because the compiler is, in general, unable to distinguish between SEEK statements that are intended to be related to a READ and those intended to be related to a WRITE.

The card record being written will be selected to the ERROR or to the AUXILIARY stackers if indicated in the particular WRITE being executed.

The WITH LOCK option WRITEs the current record and then makes it available to other programs.

This entry is only applicable to Shared Disk MCP use.

The FROM and WITH LOCK options are mutually exclusive.

ZIP.

The function of this verb is to cause the MCP to execute a control instruction contained within the operating object program.

The construct of this verb is:

```
ZIP   data-name
```

Data-name (any level) must be assigned a value equivalent to the information contained in a MCP Control Card. If the VALUE uses more than one statement card, the first character inside of the leading quote marks of the first card must be a period. Value is always ended with a period inside of the ending quote marks. ZIP may be used for programmatic scheduling of subordinate object programs contained in the Systems program library or to accomplish any of the "CC" MCP control functions as performed through the SPO or card reader.

In the statement ZIP TO-CALL-PGM2, the DATA DIVISION of the source program could contain the following entry:

```
     01   TO-CALL-PGM2   PIC X(13), VALUE IS "EXECUTE PGM2."
```

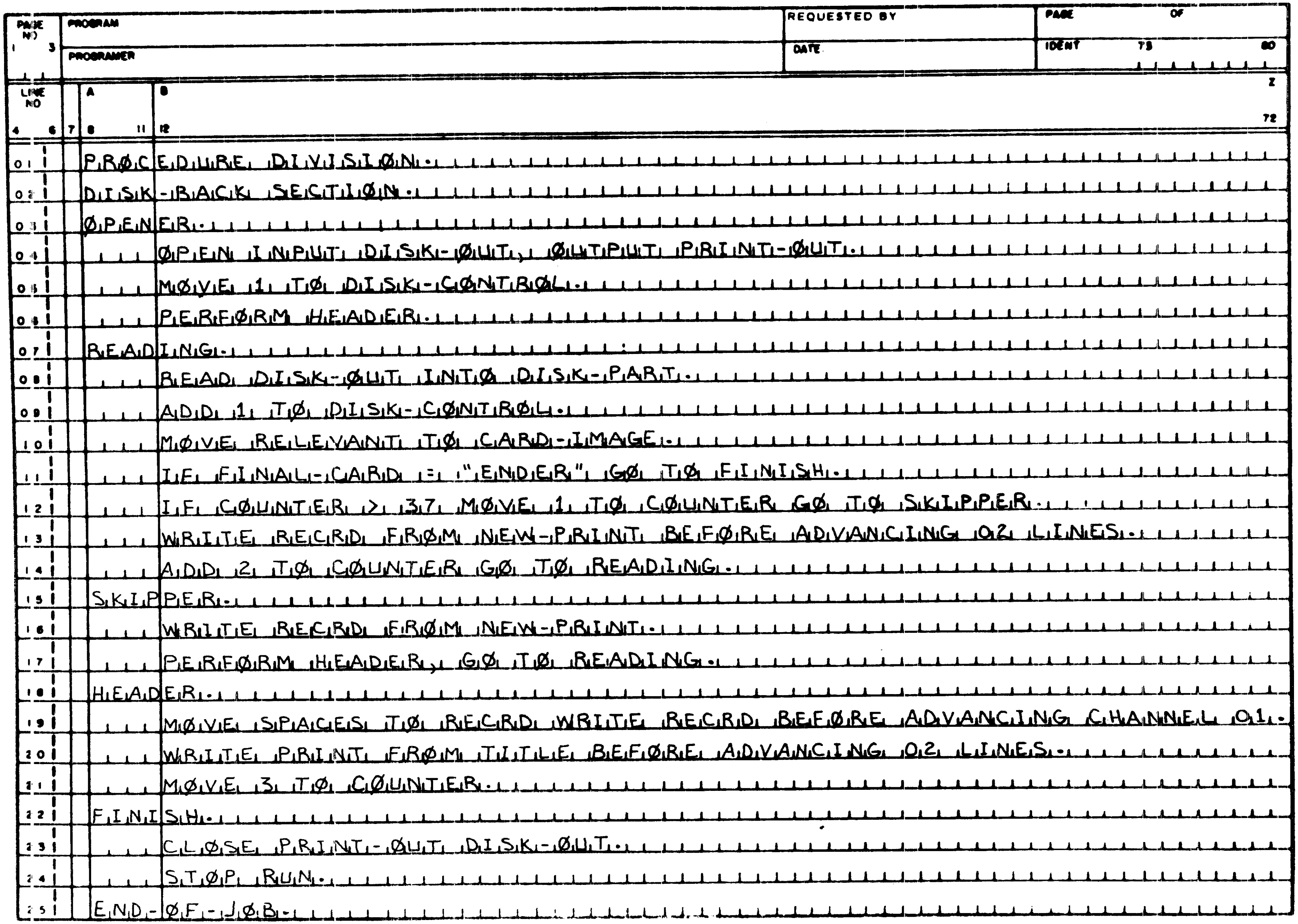
The MCP will be called upon when the object program encounters the ZIP statement and will reference data-name (TO-CALL-PGM2 in the above example) to find out which control function is being called for. Using the above example, the MCP will schedule PGM2. When the time comes and the priority for PGM2 is recognized and memory space becomes available, the MCP will retrieve PGM2 from the program library and place it in the Mix for subsequent operation. The program containing the ZIP verb will proceed to the next sequential instruction following the ZIP.

Reference must be made to the B 2500/B 3500 Master Control Programs Informational Manual for MCP Control Information formats.

CODING THE PROCEDURE DIVISION.

Figure 5-10 illustrates the manner in which the PROCEDURE DIVISION may be coded.

BURROUGHS COBOL CODING FORM



PAGE NO 1 3 | PROGRAM | REQUESTED BY | PAGE OF

PROGRAMER | DATE | IDENT 73 80

LINE NO 4 6 | 7 | A 8 11 | B 12 | Z 72

```
01 PRØCEDURE DIVISIØN.
02 DISK-BACK SECTIØN.
03 ØPENER.
04     ØPEN INPUT DISK-ØUT, ØUTPUT PRINT-ØUT.
05     MØVE 1 TØ DISK-CØNTRØL.
06     PERFØRM HEADER.
07 READING.
08     READ DISK-ØUT INTØ DISK-PART.
09     ADD 1 TØ DISK-CØNTRØL.
10     MØVE RELEVANT TØ CARD-IMAGE.
11     IF FINAL-CARD = "ENDER" GØ TØ FINISH.
12     IF CØUNTER > 37 MØVE 1 TØ CØUNTER GØ TØ SKIPPER.
13     WRITE RECRD FRØM NEW-PRINT BEFØRE ADVANCING 02 LINES.
14     ADD 2 TØ CØUNTER GØ TØ READING.
15 SKIPPER.
16     WRITE RECRD FRØM NEW-PRINT.
17     PERFØRM HEADER, GØ TØ READING.
18 HEADER.
19     MØVE SPACES TØ RECRD WRITE RECRD BEFØRE ADVANCING CHANNEL 01.
20     WRITE PRINT FRØM TITLE BEFØRE ADVANCING 02 LINES.
21     MØVE 3 TØ CØUNTER.
22 FINISH.
23     CLØSE PRINT-ØUT DISK-ØUT.
24     STØP RUN.
25 END-ØF-JØB.
```

Figure 5-10. Coding of PROCEDURE DIVISION

SECTION 6

DATA COMMUNICATIONS

GENERAL.

This section deals with the COBOL constructs of the PROCEDURE DIVISION required to activate the data communications equipment as defined by the ASSIGN to hardware-name clause.

SPECIFIC VERB FORMATS.

The specific verb formats together with a detailed discussion of the restrictions and limitations associated with each, appear on the following pages in alphabetic sequence.

NOTE

The use of any of the verbs in this section requires the presence of a version of the Data Communications MCP.

ACCEPT.

The function of this verb is to permit the entry of low-volume data from a remote SPO.

The construct of this verb is:

```
ACCEPT  data-name  FROM  { literal     }
                         { data-name-1 }
```

This statement causes the operating object program to halt and wait for appropriate data to be entered through a remote SPO. The remote SPO operator responds to an ACCEPT halt by keying in the following message:

```
? MIX-INDEX AXdata-required
```

If a blank appears between the AX and data-required, the blank character will be included in the data-stream.

When the object program executes an ACCEPT statement, the information will be transmitted from the remote SPO keyboard into memory locations assigned to the data-name. .

The number of characters transmitted must correspond to the size of the receiving data-name.

Because of the inefficiency of entering data through the remote keyboard, this teçhnqiue of data transmission should be used sparingly and solely restricted to low-volume data.

The maximum number of characters per ACCEPT statement is unlimited. ACCEPTS of greater than 60 characters must be entered thru the SPO in exact groups of 60 characters, except for the last group which can be any size up to 60 characters.

The values of literal or data-name-1 describe the name of the remote SPO from which data will be transmitted into the receiving-data-name and must be the alphanumeric name assigned to that specific remote SPO (this is the adapter ID which is specified in the MCP's UNIT CARD of the System Specification Deck). Literal-1 and data-name-1 cannot exceed six characters in length.

CLOSE

CLOSE.

The function of this verb is to cause a data communications file to be CLOSEd with or without disconnecting the remote terminal.

The construct of this verb is:

$$\underline{\text{CLOSE}}\quad \text{data-comm-file-name}\quad \left[\left\{\begin{array}{l}\text{WITH }\underline{\text{RELEASE}}\\ \underline{\text{NO}}\ \underline{\text{DISCONNECT}}\end{array}\right\}\right]$$

The specified data communications file will be CLOSEd by the initiating program.

CLOSE will keep the file assigned to the program.

CLOSE WITH RELEASE will release the remote device to the system.

If the NO DISCONNECT option is used, the file is released to the system, but the line is not disconnected.

DISABLE.

The function of this verb is to ignore input requests, either conditionally or unconditionally, from an ENABLEd device. It is also used to stop the flow of data to or from a remote device with or without disconnecting a dial line and with or without signalling a break to the operator of the remote device.

The construct of this verb is:

```
                                   ┌     ⎧ BREAK      ⎫ ┐
DISABLE file-name                  │ ON  ⎨ NO-DATA    ⎬ │
                                   └     ⎩ DISCONNECT ⎭ ┘
```

The DISABLE statement permits the object program to ignore an input request from the remote device or to interrupt a data flow.

When the NO-DATA option is used and an input request has not been received or an operation is not in process, the input request from a remote device is ignored; otherwise, the DISABLE statement will be ignored.

An unconditional disable with break will be transmitted to the remote device when the BREAK option is used. BREAK is applicable only to full duplex Data Sets.

A telephone line will be disconnected when the DISCONNECT option is used and all input requests (ringing) are ignored.

The DISABLE statement is used in the following situations:

a. After a remote device has satisfied the ENABLE and programmatic conditions arise which indicate that a WRITE is in order, the programmer must code a DISABLE file-name ON NO-DATA statement followed by an appropriate WRITE or WRITE-READ, etc.

b. Conditions arise in the process of reading from a remote device which indicates to the object program that a READ must be interrupted (one such condition might be a runaway device for which a DISABLE file-name ON BREAK would have to be initiated).

c. DISCONNECT is used with dial lines as appropriate.

DISPLAY.

The function of this verb is to provide for the printing of low-volume data, error messages, and operator instructions on a remote SPO.

The construct of this verb is:

$$\underline{\text{DISPLAY}} \quad \begin{Bmatrix} \text{literal-1} \\ \text{data-name-1} \end{Bmatrix} \quad \left[\begin{Bmatrix} \text{literal-2} \\ \text{data-name-2} \end{Bmatrix} \quad \ldots \right]$$

$$\underline{\text{UPON}} \quad \begin{Bmatrix} \text{literal-3} \\ \text{data-name-3} \end{Bmatrix}$$

Data-name-1 and literal-1 (and their associated series) are specified as being the area within an object program from which data is to be transmitted to a receiving remote SPO.

The DISPLAY statement causes the contents of each operand to be transmitted from the MCP SPO queue to ensure that an operational program is not delayed while the remote SPO message is being printed.

Literal-1 and literal-2 may consist of any figurative constant, except ALL.

If a figurative constant is specified as literal-1 and/or literal-2, only a single character of the figurative constant will be DISPLAYed.

Data-name-1 and data-name-2 may be subscripted and can be PICTUREd as being composed of COMPUTATIONAL or DISPLAY data.

An infinite amount of characters may be contained in a literal, or data-name to be DISPLAYED. The compiler will automatically supply the proper number of carriage returns and line feeds into the object program, as may be appropriate.

The DISPLAY series (15 maximum) option will cause the literals or data-names to be printed one after another. If required, the compiler will cause automatic carriage returns and line feeds to be executed by the object program for information extending to continued lines of print. The compiler will not supply partial word formatting at the end of a line. Data will be printed as received, until a carriage return and line feed is inserted by the object program, thus, words of data can be split between continued lines of output.

The value contained in literal-3, or data-name-3, describe the name of the remote SPO to which data will be transmitted and <u>must</u> be the alphanumeric name assigned to that specific remote SPO as defined in the MCPs UNIT CARD of the Systems Specification Deck. Literal-3 and data-name-3 cannot exceed six characters in length.

ENABLE.

The function of this verb is to recognize input inquiry requests from a remote device, to disconnect the telephone line for dial lines, recognize a ringing signal, or, recognize an inquiry (ENQ) from an appropriate remote device.

The construct of this verb is:

```
ENABLE file-name [PROCEED TO paragraph-name]
```

File-name must have been OPENed before an ENABLE can be executed.

Once the file-name has been ENABLEd, a WAIT statement may be used to suspend processing until appropriate response is received. Reference must be made to the WAIT verb, as contained in this section, for proper definition.

ENABLE allows the device to establish a connection with the B 2500/ B 3500 System by depressing the Inquiry Key (ENQ) if the device is connected on leased lines or by dialing the systems telephone number if the device is on dialed lines.

Paragraph-name must be in the non-overlayable program segment or in the same segment as the WAIT.

The ENABLE statement will cause dial telephone lines to be disconnected and will recognize input requests from the device in the form of a telephone ringing signal.

The ENABLE statement for leased lines will recognize input requests from the remote device in the form of an inquiry (ENQ).

FILL.

The function of this verb is to initiate a specified type of I-O and allow a program to run without waiting for an I-O to be completed. It is useful when a program is handling more than one remote device or when a program does not require the input data to continue processing.

The construct of this verb is:

```
FILL file-name [NO-TIME-OUT] [START-TEXT] [DIAL] [END-TEXT]

                                         ( WRITE                )
                                         | READ                 |
[POLL] [VOICE] [TONE] [STREAM] WITH     <  WRITE-READ            >
                                         | WRITE-TRANS-READ     |
                                         ( WRITE-READ-TRANS     )

[PROCEED TO paragraph-name]
```

FILL causes the operation to be initiated. The program must accomplish a READ to move the required data to the record-area.

If the option NO-TIME-OUT WITH WRITE-READ is used, the time-out feature will be inhibited on the READ of the WRITE-READ clause.

If the option START-TEXT WITH READ is used, the first code received is considered text and is used in generating the longitudinal redundancy check character. The start-of-text character will not be utilized.

If the option DIAL WITH READ is used, a dial number is accessed from memory starting at the 01 level entry of the record-description for the file-name specified. The dial numbers must be declared USAGE COMPUTATIONAL and the dial number field must be terminated by the binary control code of 1100 (undigit 12) which can be represented in B 2500/B 3500 COBOL as an undigit literal @C@. The total number of digits comprising the dial number must be even, and a FILLER digit with a zero value must be inserted after the undigit literal if necessary. The rest of the record-description for file-name will describe the input data from the control code or FILLER, whichever is applicable, to the end of the record description.

If the option END-TEXT WITH WRITE-READ is used, the control code denoting End-of-Text (ETX) is not transmitted. The ETX function is ignored and the longitudinal redundancy check (LRC) character is not generated or sent. This option is used for polling operations only.

If the option VOICE WITH WRITE-READ is used, the voice response adapters are ENABLEd automatically. Characters received from memory are sent to the voice responder and are used as voice-track addresses. The resulting signals from the voice responder are sent over the line.

If the option TONE WITH WRITE-READ is used, the tone leads on the Tone Data Sets are activated. The character "A" produces a 1017 hz (hertz) tone and the "B" produces a 2025 hz (hertz) tone. Any other character produces silence. The tones will continue for 300 milliseconds per character sent.

The POLL option is used only with WRITE-READ. The output area will consist of a series of contiguous polling or selection sequences. A WRITE-READ will be executed with the first such sequence; if a message other than a negative response is returned, the entire operation terminates with the control character at the end of the message, otherwise, the negative response is discarded and another WRITE-READ is automatically initiated using the next sequence. When only negative responses are returned, the operation is terminated by two successive ending characters. If a message is returned, it will be written in the area following the sequence which received the response, thus overlaying the successive sequences.

If the STREAM option is specified, the information is transferred into ascending memory locations starting at the 01 level of the record description for the specified file-name. See the STREAM option of the READ statement in this section for further information.

FILL statements normally are initiated to several remote devices and are followed by a WAIT statement.

INTERROGATE.

The function of this verb is to obtain a result descriptor representing the operational status of a remote device.

The construct of this verb is:

```
INTERROGATE [END-TEXT] file-name INTO data-name
```

Data-name, when END-TEXT is not specified, must be defined as an elementary item with a PICTURE 9(16) COMPUTATIONAL and must be REDEFINEd to make programmatic reference to each element within the result descriptor.

The COBOL result descriptor contains 16 digits which indicate conditions that occurred during an I/O operation. The digits of information within the result descriptor are assigned the following meanings when turned "ON".

Digit Number	Meaning
1	Operation complete.
2	Exception condition.
3	Not ready local (single-line). (Multi-line if during operation).
4	Data error.
5	Abandon call retry (ACR).
6	Cancel complete
7	End-of-transmission (EOT).
8	Attempt to exceed maximum address.
9	Time out.
10	Memory parity error.
11	Write error.
12	Carrier loss.
13 through 16	Reserved.

Digit Number	Meaning
4 and 5	Data loss.
6 and 7	Break detected.

ON status is indicated by a value of 1. OFF status is indicated by a value of zero. Digits are independent of one another and can reflect varied combinations.

Explanation of the result descriptor digits is as follows:

Digit Number	Status
1	Always ON if the attempted operation was completed.
2	Will be ON if any combination of three through 16 are ON. This is the test position to see if any exception exists. If this position is ON by itself, a partial complete condition exists due to the use of READ STREAM MODE and will not occur in any other situation.
3	Will be ON if the single line control or the local Data Set is not ready and the operation will be terminated. For multi-line control the digit is set ON in the channel result descriptor unless it occurs during an operation, in which case it is set ON in the adapter result descriptor.
4	If a data error (message or character parity) occurs, a READ operation continues until terminated in a normal manner. The phone line is not disconnected. Attempts to exceed maximum address, time out, End-of-Text (ETX), or End-of-Transmission (EOT) can also occur.
4 and 5	If data loss (missed memory access or MLC cycle), a READ operation continues until

Digit Number — Status

terminated in a normal manner. Attempts to exceed maximum address, time out, End-of-Text (ETX), or End-of-Transmission (EOT) can also occur. The phone line is not disconnected. A WRITE operation is terminated immediately and position 11 is set.

5 — If an abandon call retry condition exists, this position will be set ON and the telephone line is disconnected.

6 — If a cancel complete condition exists, this position will be set ON and CANCEL is initiated.

6 and 7 — If a break is detected, these positions will be set ON for a WRITE operation only and the operation is immediately terminated. The telephone line is not disconnected.

7 — If the End-of-Transmission exists, this position is set ON and the telephone line is disconnected.

8 — If an attempt to exceed maximum address exists, a READ operation will initiate a time out and wait for a control code denoting End-of-Text (ETX). This position will be set ON if an End-of-Text (ETX) is received before time out. This position along with position 7 will be set ON if an (EOT) is received before time out. This position and position 9 will be set ON if time out occurs without ETX or EOT. A WRITE operation is immediately terminated and this position along with position 11 is set ON. The telephone line is disconnected in each case.

Digit Number	Status
9	If time out exists, this position is set ON and the telephone line is not disconnected. Time out occurs on READ instructions only.
10 and 11	If a memory parity error exists, these positions are set ON and the telephone line is not disconnected. Memory parity error occurs only during a WRITE operation, and will immediately terminate the operation.
12	A READ operation continues until terminated in a normal manner. The phone line is not disconnected. Attempt to exceed maximum address, time out, End-of-Text (ETX) or End-of-Transmission (EOT) can also occur.
13	Used in multi-line control to check end-of-stream, when operating in stream mode.
14 and 16	Reserved.

If the END-TEXT option is specified, data-name will contain the number of characters transmitted to and/or from the current buffer of the file. Counting begins when the descriptor is initiated and continues until it is complete. Data-name must be defined as PC 9(6) COMPUTATIONAL. If the I-O is not complete on the current buffer, the INTERROGATE END-TEXT is ignored.

READ.

The function of this verb is to load data from a remote device into ascending memory locations beginning with the location specified by the 01 level of the record-description of a file.

The construct of this verb is:

```
READ file-name [INTO record-name] [NO-TIME-OUT] [START-TEXT]

 [STREAM] [DIAL] [AT END any statement]

 [{ OTHERWISE }      any statement ]
 [{ ELSE      }                    ]
```

Loading will continue until an ending code such as End-of-Transmission (EOT), End-of-Text (ETX) or End-of-Block (EOB) is detected, or until the buffer is filled.

The time out feature is inhibited on the READ if the NO-TIME-OUT option is used.

If the START-TEXT option is used, the first code received is considered text and is used in generating the Longitudinal Redundancy Check (LRC) Character. The start of text character will not be used.

If the STREAM option is used, the information is written into ascending memory locations starting at the 01 level of the record-description for the file-name specified. The record-description entry must define at least 200 digits (100 bytes). A control code denoting End-of-Text (ETX) will terminate the operation. The use of a FILLER after the End-of-Text (ETX) character in the record-description will be required to ensure that the End-of-Text (ETX) control code will not be the last position in that entry.

See the DIAL WITH READ option of the FILL statement for the DIAL requirements.

If the AT END option is used, an End-of-Transmission (EOT) character received will cause the program to accomplish the indicated actions. NEXT SENTENCE is implied in the absence of an AT END statement.

WAIT.

The function of this verb is to suspend an object program until an ENABLE or FILL statement is initiated and/or to suspend an object program for a specified number of seconds.

The construct of this verb has two options which are:

Option 1:

```
WAIT
```

Option 2:

```
WAIT UNTIL  { literal   }
            { data-name }
```

Literal reflects the number of seconds that the object program is to be suspended. The maximum WAITing period is 23 hours, 59 minutes, and 59 seconds (86,399 seconds).

Option 1 is used in conjunction with the ENABLE or FILL statements if a WAITing period is desired. It normally follows either statements.

In Option 1, FILL statements are normally initiated to several lines and are followed by a WAIT statement. This statement would not ordinarily be used when only one device is involved, as a FILL statement by itself will affect the same action.

In Option 2, the object program is suspended until an ENABLE or FILL statement is initiated or until the number of seconds have elapsed, whichever comes first. Data-name must be described as PC 9(5) COMPUTATIONAL.

The following rules apply when WAIT is used in conjunction with the ENABLE of FILL constructs:

a. If an ENABLE and/or FILL statement contains a PROCEED TO paragraph-name and either statement comes "true", the object program will be reinstated at the appropriate PROCEED TO paragraph-name.

b. If the PROCEED TO option is omitted and an ENABLE and/or FILL statement comes "true", the object program will be reinstated at the next instruction following the WAIT UNTIL statement. This rule does not apply to Option 1.

c. If a PROCEED TO option is omitted, the object program will not know how it got to the next instruction. That is: Did a FILL or ENABLE come "true" or did the WAIT UNTIL time expire? An INTERROGATE of all ENABLEd or FILLed files will have to be performed to determine the answer.

WRITE.

The function of this verb is to pass data to a remote device from ascending memory locations beginning at the 01 level of the pertinent record-description.

The construct of this verb is:

```
WRITE record-name-1 [FROM record-name-2] [END-TEXT]

 [START-TEXT] [STREAM] [DIAL] [VOICE] [TONE]  [AT END

 any statement]     [ { OTHERWISE }    any statement  ]
                    [ { ELSE      }                   ]
```

Data will be passed until a control code denoting End-of-Transmission (EOT) or End-of-Text (ETX) is detected in record-name-1.

If the END-TEXT option is used, the End-of-Transmission (EOT) control code is not transmitted, the End-of-Transmission function is ignored, and the Longitudinal Redundancy Check Character (LRC) is not generated or sent.

If the START-TEXT option is used, the first code sent is considered text and is used in generating the LRC. The start of text function is automatically preset.

If the STREAM option is used, the information is passed from ascending memory locations starting at the 01 level of record-name-1 See the STREAM option of the READ statement for further information.

If the DIAL option is used, the dial number is accessed from memory starting at the 01 level entry of record-name-1. Information to be passed to a remote device will begin at the level entry following the @C@ control code or the FILLER digit, whichever is applicable.

See the DIAL WITH READ option of the FILL statement for further DIAL requirements.

If the VOICE option is used, the voice response adapters are ENABLEd automatically. Characters are passed from memory to the voice responder and are used as voice track addresses. The resulting signals from the voice responder are sent over the line.

If the TONE option is used, the tone leads on the Tone Data Sets are activated. Characters received from memory by the I/O adapter are sent to the tone leads as follows:

a. "B" characters to the 2025 hz lead.
b. "A" characters to the 1017 hz lead.
c. All others to the silent lead.

The tones or silence will continue for 300 milliseconds per character sent.

Reference the FILL and WAIT constructs, this section, for requirements of FILL WITH WRITE and FILL WITH WRITE and WAIT.

WRITE-READ.

The function of this verb is to pass data to a remote device from memory locations and, when successfully completed, to cause data to be read from the remote device and passed to appropriate memory locations.

The construct for this verb is:

WRITE-READ record-name-1 [FROM record-name-2] [NO-TIME-OUT]
[END-TEXT] [START-TEXT] [VOICE] [DIAL] [TONE] [POLL] [STREAM]
[AT END any statement]

Data will be passed to the remote device from ascending memory locations starting at the 01 level of record-name-1 and will continue until a control code denoting End-of-Transmission (EOT) or End-of-Text (ETX) is detected. A READ will then be initiated on the remote device and the data will be passed to ascending memory locations beginning with the location immediately following the End-of-Transmission (EOT) or End-of-Text (ETX) control code which terminated the WRITE and will continue until an End-of-Text (ETX) control code from the remote device is encountered. Each portion of the message being written and read must be terminated by an End-of-Transmission (EOT) or End-of-Text (ETX) control code.

NO-TIME-OUT, see the READ statement.

END-TEXT, see the WRITE statement.

START-TEXT, see the READ statement.

VOICE, see the WRITE statement.

DIAL, see the WRITE statement.

TONE, see the WRITE statement.

POLL, see the WRITE statement.

STREAM, see the WRITE statement.

If the AT END option is used, an End-of-Transmission control code received will cause the program to accomplish the indicated actions.

NEXT SENTENCE is implied in the absence of an AT END statement.

Reference the FILL and WAIT constructs, this section, for requirements of FILL WITH WRITE and FILL WITH WRITE and WAIT.

WRITE-READ-TRANS.

The function of WRITE-READ-TRANSparent is to pass data to a remote device (normally a computer) from memory locations and, when successfully completed, to cause data to be read from the remote device and passed to appropriate memory locations and terminating at the end of the record-description without passing an End-of-Transmission (EOT) or End-of-Text (ETX) control code.

The format of WRITE-READ-TRANS is:

```
WRITE-READ-TRANS record-name-1 [FROM record-name-2]

 [NO-TIME-OUT] [DIAL]  [AT END any statement]
```

Data will be passed to the remote device from ascending memory locations starting at the 01 level of record-name-1 and will continue until a control code denoting End-of-Transmission (EOT) or End-of-Text (ETX) is detected. A READ will then be initiated on the remote device and the data will be passed to ascending memory locations beginning with the location immediately following the End-of-Transmission (EOT) or End-of-Text (ETX) control code which terminated the WRITE and will continue until the end of record-name-1. The Attempt To Exceed Maximum Address in the Result Descriptor will not be turned ON when the end of record-name-1 is reached.

NO-TIME-OUT, see the READ statement.

DIAL, see the WRITE statement.

The AT END option will be initiated when NEXT SENTENCE is implied in the absence of an AT END statement.

This statement is normally used for remote computers to the system.

WRITE-TRANS-READ.

The function of this verb is to pass data to the remote device (normally a computer) until the end of the record description is reached and, when successfully completed, to cause data to be passed from the remote device to memory locations starting at the end of the record-description and continuing until an End-of-Transmission (EOT) or End-of-Text (ETX) control code is detected.

The construct of this verb is:

```
WRITE-TRANS-READ  record-name-1 [FROM record-name-2]

 [NO-TIME-OUT] [DIAL] [AT END any statement]
```

The READ portion of this statement will continue passing data until an End-of-Transmission (EOT) or End-of-Text (ETX) control code is detected but will cut off the flow when location ending record-name-1 address + 199 is reached.

NO-TIME-OUT, see the READ statement.

DIAL, see the WRITE statement.

The AT END option will be initiated when NEXT SENTENCE is implied in the absence of an AT END statement.

This statement is normally used for remote computers to the system.

The end of the record-area may be programmatically altered by use of the ACTUAL KEY clause in the FILE-CONTROL paragraph. The ACTUAL KEY data-name must be defined as PC 9(6) COMPUTATIONAL. The value contained there-in will be used when the I-O is initiated to determine the number of characters in the record area. The ending address of the result descriptor will be adjusted accordingly.

SECTION 7

CODING FORM

GENERAL.

The coding form, which provides a standard method for describing COBOL source programs, has been defined by CODASYL specifications and common usage. The B 2500/B 3500 COBOL Compiler accepts this standard coding format, but also allows certain departures from the standard, at the user's discretion.

The same coding form is used for all four divisions of the source program. The four divisions must appear in the following order: IDENTIFICATION DIVISION, ENVIRONMENT DIVISION, DATA DIVISION, and PROCEDURE DIVISION. Each division must be written according to the rules for the coding form.

The rules for spacing given in the following discussion of the coding form take precedence over all other rules for the coding form.

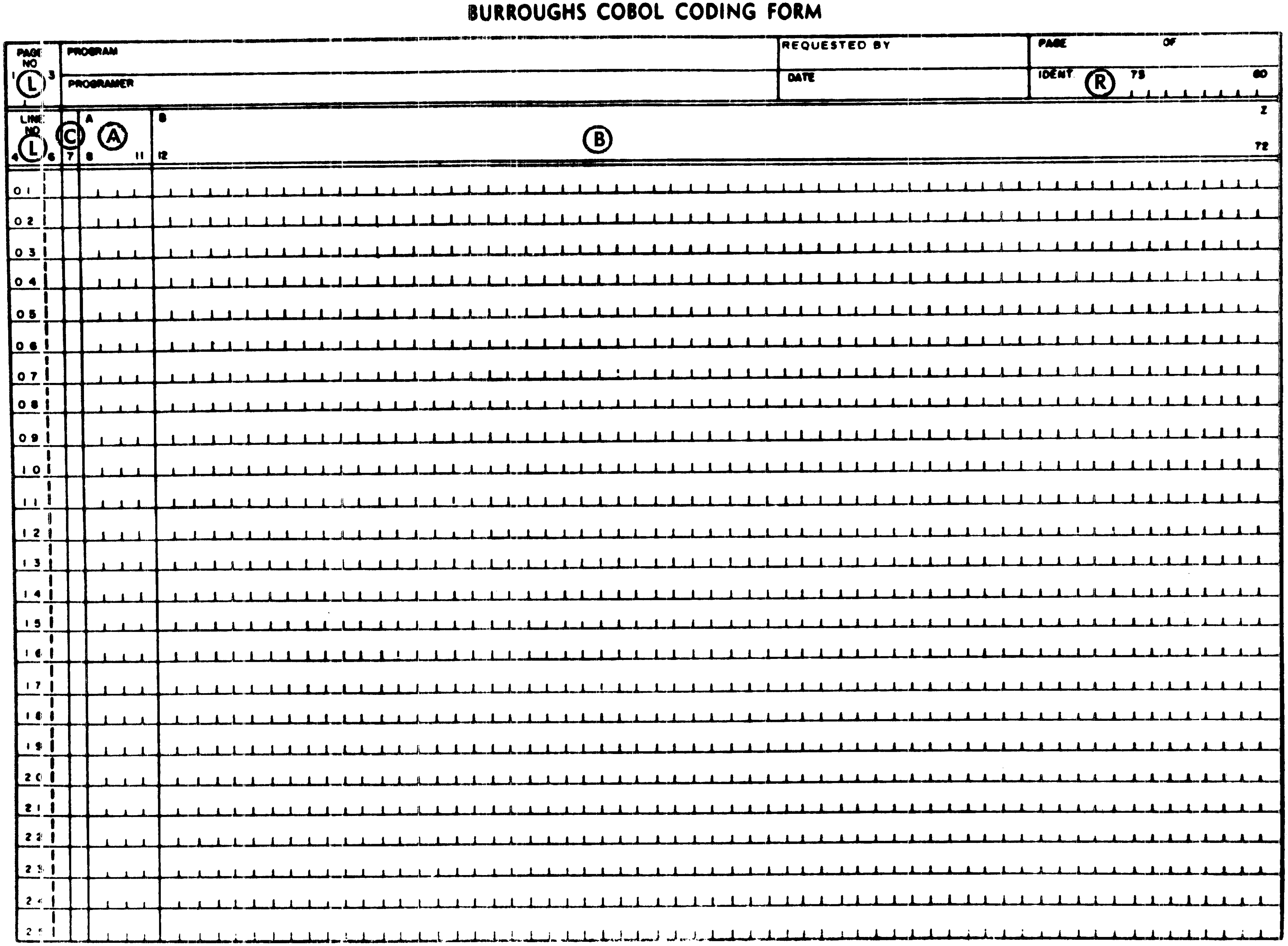
CODING FORM REPRESENTATION.

The coding format for a line is represented in figure 7-1. The digits designate columns.

1 2 3 4 5 6	7	8 9 10 11	12 13 ... 72	73 ... 80
Sequence Number Area	Continuation Area	Area A	Area B	Identification Area
MARGIN L	MARGIN C	MARGIN A	MARGIN B	MARGIN R

Figure 7-1. Coding Format for a Source Line

Figure 7-2 provides a sample of the COBOL Coding Form.



BURROUGHS COBOL CODING FORM

PAGE NO 1 L 3	PROGRAM PROGRAMER	REQUESTED BY DATE	PAGE OF IDENT. R 73 80
LINE NO 4 L 6	C 7	A A 8 11	B B 12	Z 72
01				
02				
03				
04				
05				
06				
07				
08				
09				
10				
11				
12				
13				
14				
15				
16				
17				
18				
19				
20				
21				
22				
23				
24				
25				

Figure 7-2. Sample Coding Form

SEQUENCE NUMBERS (COLUMNS 1-6).

The sequence number field may be used to sequence the source program cards. Normally, numeric sequence numbers are used; however, the COBOL Compiler allows any combination of characters from the allowable character set. The compiler generates a warning message during compilation time if a sequence error (other than ascending) occurs.

CONTINUATION INDICATOR (COLUMN 7).

A hyphen in the Continuation Area of the continuation line indicates that the first character in Area B is the continuation of a word or a literal from the previous line. If a hyphen does not occur in the Continuation Area, the word or literal starting in Area B is not a continuation of an entry which started on the previous line and is separated from the previous entry with a space.

An asterisk (*) indicates that the source line is for documentation purposes only and can appear anywhere within the source program. Continuation of following lines is denoted by an asterisk in column 7 of the continued data. All entries of this type are free form from Area A through Area B.

A slash (/) indicates that the source line is for documentation purpose only and that a skip to the head of a new page is required during the listing phase of the compiler output.

The letter L followed by a "library-name" entry, will cause all succeeding source card data to be placed into the COBOL Library File during compilation. Termination of the action takes place when an L card is encountered followed by spaces.

CONTINUATION OF UNDIGIT LITERALS.

When an undigit literal is continued from one line to another, a hyphen is placed in the Continuation Area (column 7) of the continuation line, but the at sign (@) is not placed in the first character position of Area B (column 12). The continuation of the undigit literal commences in column 12 of Area B.

CONTINUATION OF NON-NUMERIC LITERALS.

When a non-numeric literal is continued from one line to another, a hyphen is placed in the Continuation Area (column 7) of the continuation line and a quotation mark must be the first non blank position of Area B. The continuation of the non-numeric literal commences immediately following the quotation mark. All spaces at the end of the continued line and any spaces following the quotation mark of the continuation line and preceding the final quotation mark of the non-numeric literal are considered part of the literal.

CONTINUATION OF WORDS AND NUMERIC LITERALS.

When a word or numeric literal is continued from one line to another, a hyphen is placed in the Continuation Area of the continuation line. This indicates that the first character of Area B of the continuation line is to follow the last non-blank character of the continued line without an intervening space.

Figure 7-3 illustrates the use of the Continuation Indicator for both non-numeric literals and other word entries.

DIVISION HEADER.

The Division Header must be the first line of a division coding format. The Division Header starts in Area A with the division name, is followed by a space, then the word DIVISION, and then a period. No other text may appear on the same line as the Division Header.

SECTION HEADER.

The name of a section starts in Area A of any line except the first line of a division coding format. It is followed by a space, then the word SECTION, and then a period. In the PROCEDURE DIVISION, an option may be exercised by which the word SECTION would be followed by a space followed by a priority number. As above, the priority number would be followed by a period. No other text may appear on the same line as the Section Header except in the declarative portion of the PROCEDURE DIVISION. In this case, the USE and COPY sentences may begin on the same line as the Section Header. A section consists of paragraphs in the ENVIRONMENT and PROCEDURE DIVISIONs and data description entries in the DATA DIVISION. Paragraph names, but no section names, are permitted in the IDENTIFICATION DIVISION.

PARAGRAPH NAMES AND PARAGRAPHS.

The name of a paragraph starts in Area A of any line following the first line of a division coding format and ends with a period. A paragraph consists of one or more successive sentences. The first sentence in a paragraph begins in Area B of either the same line as the paragraph name or any succeeding line. Successive sentences either begin in Area B of the same line as the preceding sentence or in Area B of the next line. A sentence consists of one or more statements, followed by a period.

DATA DIVISION ENTRIES.

Each DATA DIVISION entry begins with a level indicator or a level number, followed by one or more spaces, followed by the name of a data item, followed by a sequence of independent clauses described in the DATA DIVISION. Each clause, except the last clause of an entry, may be terminated by a semicolon or comma. This last clause is always terminated by a period.

There are two types of DATA DIVISION entries: those which begin with a level indicator and those which begin with a level number. A level indicator is an FD. In those DATA DIVISION entries which begin with the level indicator FD, the level indicator begins in Area A followed by a space and then by its associated file name and appropriate descriptive information and terminated with a period.

DATA DIVISION entries that begin with level numbers are called data description entries. A level number may be one of the following set: 01 through 49, 66, 77, and 88. Level numbers are written either as a space followed by a digit or a zero followed by a digit. At least one space must separate the level number from the word that follows it.

Level numbers 01, 66 and 77 should be coded in Area A. Other level numbers should be coded in Area B. Each successively higher level number should be indented four positions. This makes the coding easier to follow, and structure is readily apparent. Using odd numbered level numbers permits easy patching of record descriptions.

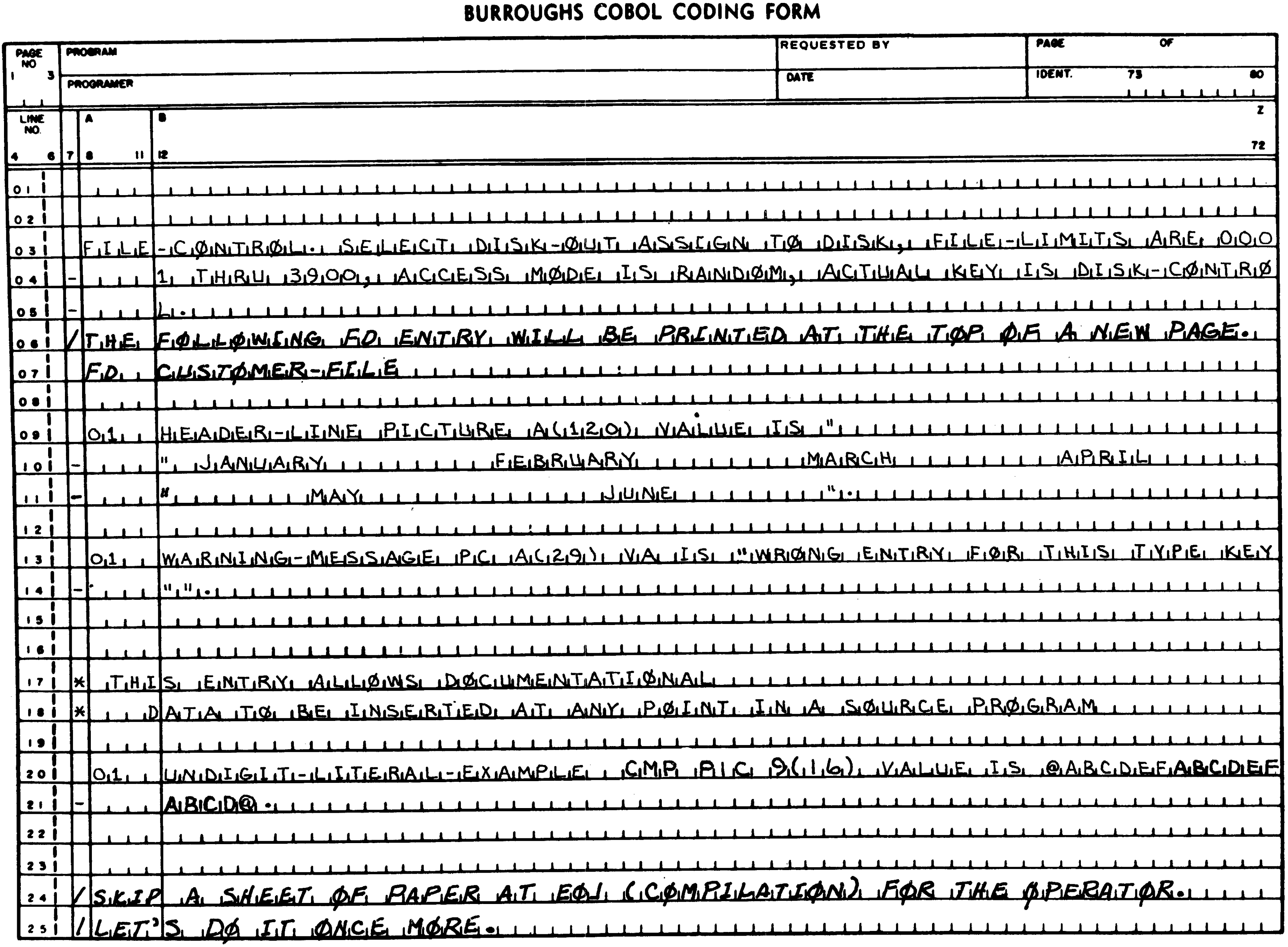
Coding repetitive information in the same columns makes keypunching easier; such as, PIC in columns 36-37, VALUE columns 52-56.

For example:

```
01   INPUT-RECORD.
     03  AMOUNT              PC  9(5)V99.
     03  AMOUNT-OUT          PC  9(5)V99.
     03  FACTORS             PC  9(3)V9(5).
     03  PERCNT              PC  V999.
     03  NAME-CITY           PC  X(10).
     03  CODR                PC  XX.
     03  DATER.
         05  MONTH           PC  99.
         05  DAY             PC  99.
         05  YEAR            PC  99.
             88  CUR-DECADE                  VA
                 60 THRU 69.
     03  FILLER              PC  X(33).
66   IN-DATE RENAMES MONTH THRU YEAR.
```

NOTE

The above 88 level is continued on the following line, however, a dash in column 7 must not appear, inasmuch as the compiler continues to scan the following line in an effort to satisfy the VAlue requirement.



BURROUGHS COBOL CODING FORM

PAGE NO 1 3 | PROGRAM | REQUESTED BY | PAGE OF
PROGRAMER | DATE | IDENT. 73 80

```
LINE NO.  7  A    B
4      6     8 11 12                                                            Z 72
01
02
03          FILE-CONTROL. SELECT DISK-ØUT ASSIGN TØ DISK, FILE-LIMITS ARE 000
04       -       1 THRU 3900, ACCESS MØDE IS RANDØM, ACTUAL KEY IS DISK-CØNTRØ
05       -       L.
06       /  THE  FØLLØWING FD ENTRY WILL BE PRINTED AT THE TØP ØF A NEW PAGE.
07          FD   CUSTØMER-FILE
08
09          01   HEADER-LINE PICTURE A(120) VALUE IS "
10       -       " JANUARY          FEBRUARY         MARCH           APRIL
11       -       "       MAY          JUNE          ".
12
13          01   WARNING-MESSAGE PC A(29) VA IS "WRØNG ENTRY FØR THIS TYPE KEY
14       -       "".
15
16
17       *   THIS ENTRY ALLØWS DØCUMENTATIØNAL
18       *     DATA TØ BE INSERTED AT ANY PØINT IN A SØURCE PRØGRAM
19
20          01   UNDIGIT-LITERAL-EXAMPLE  CMP PIC 9(16) VALUE IS @ABCDEFABCDEF
21       -       ABCD@.
22
23
24       /  SKIP A SHEET ØF PAPER AT EØJ (CØMPILATIØN) FØR THE ØPERATØR.
25       /  LET'S DØ IT ØNCE MØRE.
```

Figure 7-3. Sample Coding Showing Continuation of Lines, Special Remarks, and Actions

DECLARATIVES.

The key word DECLARATIVES and the key words END DECLARATIVES that precede and follow the Declaratives portion of the PROCEDURE DIVISION, respectively, must each appear on a line by itself. Each must begin in Area A and be followed by a period.

PUNCTUATION.

The following rules of punctuation apply to the writing of COBOL programs for the B 2500/B 3500:

a. A sentence is terminated by a period. A period may not appear within a sentence unless it is within a non-numeric literal or is a decimal point in a numeric literal or is in a PICTURE.

b. Two or more names in a series must be separated by a space or a comma.

c. Semicolons are used for readability and are never required. The semicolon is used for separating statements within a sentence or clauses within data description entries.

d. The reserved word THEN is also used for readability and can be used to separate two statements within a sentence. It can also be used between the condition and the first statement within an IF statement. For example:

```
IF .... THEN .... THEN .... ELSE ....
```

e. A space must never be imbedded in a name; hyphens may be used instead. However, a hyphen may not start or terminate a name. For example:

PRODUCTION-PERIOD	is a good data-name, section-name, or paragraph-name.
-PRODUCTION-PERIOD	<u>or</u> -PRODUCTION-PERIOD- <u>or</u> PRODUCTION-PERIOD- are all bad entries.

SECTION 8
COBOL COMPILER CONTROL

GENERAL.

There are currently two versions of the B 2500/B 3500 COBOL Compiler: COBOLL (30KB) and COBOL (17KB). The COBOLL (or COBOL) Compiler, in conjunction with the MCP, allows for various types of actions during compilation and is explained in the text that follows.

COMPILATION CARD DECK.

Control of the COBOL Source Language input is derived from presenting the Compilation Card Deck, illustrated in figure 8-1, to the MCP.

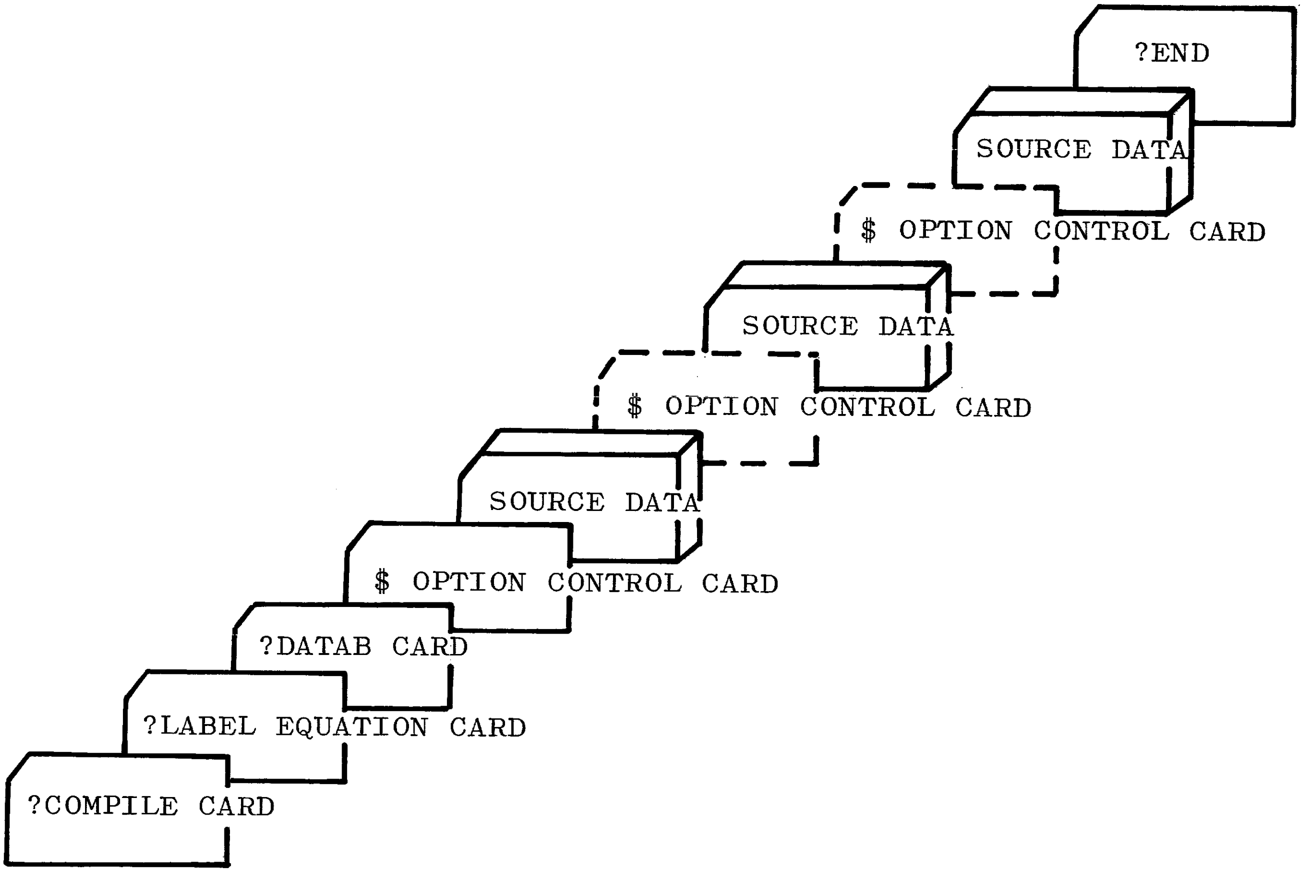


Figure 8-1. Compilation Card Deck

The Compilation Card Deck is comprised of several cards; these cards, along with a detailed discussion of their function are presented in the paragraphs that follow.

?COMPILE CARD.

The first input control card instructs the MCP to call-out either the COBOLL or COBOL Compiler and to complete the indicated program-name P-N using one of the following options:

a. To compile and run the resultant object program, the card is coded:

    ?COMPILE P-N WITH COBOLL (or COBOL)

b. To compile for a syntax check only, the card is coded:

    ?COMPILE P-N WITH COBOLL SYNTAX (or COBOL)

c. To compile and place the resultant object code into the Systems Library, the card is coded:

    ?COMPILE P-N WITH COBOLL LIBRARY (or COBOL)

d. To compile and place the resultant object code into the Systems Library, and then run the object program, the card is coded:

    ?COMPILE P-N WITH COBOLL SAVE (or COBOL)

e. To give the compiler more core space to operate in, the card is coded:

    ?COMPILE P-N WITH COBOLL CORE nnnnnn (or COBOL)

   The nnnnnn entry must be larger than the compiler size being used.

The absence of the ?COMPILE card will cause the System Operator to manually execute one of the above options through the SPO, using the MCP's CC notation in place of the question mark (?).

MCP LABEL CARD.

The second control card is the MCP LABEL card and is formatted in either of the following forms:

a. ?DATA CARD (indicates EBCDIC source language input).
b. ?DATAB CARD (indicates BCL source language input).

The absence of the MCP LABEL card will cause the message,

    **NO FILE file-name program-name = mix-index to be displayed on the SPO. The System Operator will not know the proper IL message to give the MCP (because of the options involved), without being given specific instructions by the programmer.

$ OPTION CONTROL CARD.

The third card is the COBOLL (or COBOL) Compiler Option Control card ($ sign in column 7). This card is used to notify the compiler as to which options are required during the compilation. If this card is omitted, $ CARD LIST is assumed. The format of the Compiler Option Control card is as follows:

```
        $ option (option) ...
```

The options available for both COBOLL and COBOL Compiler Option Control cards are as follows:

a. CARD - input is from the source language cards or paper tape.

b. CODE - list object code from the point of insertion.

c. MCPB - inhibits the COBOL Compiler from generating object code for blocking and unblocking of input and/or output records. The omission of this option will generally cause the object program to operate faster when reading and writing blocked logical records. However, the user will forfeit a small portion of core, for each file, to obtain the benifits of an increase in speed. The COBOL blocking and unblocking intrinsic will only be used by the compiler when the MCPB Compiler Option is not present and:

   1) Files are declared as containing fixed-length records without a FILE-LIMITS clause.

   2) Records are contained on magnetic tape or reside on disk as SEQUENTIAL files.

   The MCP option can be inserted into the source program at a point just prior to every FD for which MCP blocking or deblocking is desired. Note that this option card must contain all other options required to control the compilation from point of insertion.

d. TAPE - input is from a SOurce Language Tape (file-ID is SOLT) with correction/inserts and change cards included from either the card reader, paper tape, or magnetic tape. CARD is assumed by the compiler if TAPE is omitted. The SOLT tape is created 80 character records, blocked five logical records per block. A user created SOLT may contain multiple files of source language programs.

e. NEWT - creates a new SOLT from cards or from an old SOLT plus pertinent changes. The presence of a NEWT in the control card does not require that a magnetic tape be available when another card immediately follows it without the NEWT option.

f. LIST - creates a double-spaced output listing of the source language input, with error messages, where required.

g. LST1 - same as LIST, except that the listing will be single spaced.

h. SUPR - suppresses warning messages.

i. SPEC - negates LIST or LST1 if syntax errors occur. If both SPEC and SUPR are specified and no syntax errors occur, all printing is suppressed including the final summary.

j. BLNK - causes all cards with columns 7-72 blank to be automatically purged. A subsequent control card without BLNK will turn off this option.

k. Non-numeric literal - is inserted in columns 73-80 of all following card images when creating a new tape (NEWT) and/or printing. This option can be turned off or changed by a subsequent control card.

l. +nnnnnn - re-sequencing increment of source language input in the output list and a new SOLT if applicable. Re-sequencing is re-initialized by each subsequent control card, or turned off if a subsequent control card does not contain this option.

m. nnnnnn - re-sequencing starting number of Source Language input. Re-sequencing is re-initialized by each subsequent control card, or turned off if a subsequent card does not contain this option.

The COBOLL compiler contains all of the above options, plus the following:

a. JAPN - causes the output listing to be compressed and to start in column 37 of the print line.

b. SKIP nn - used by the programmer to specify the number of lines on a page for a COBOL source print-out. The letters nn designate the number of lines desired. If this option is omitted from the $ CARD, channel 12 is used.

c. XREF - when coded in the $ CARD, a cross-reference of the compiled program is printed out, after compilation by the COBXR Program which is on the SYSTEM tape. This cross-reference is not generated by the Cross-reference Program, CBXRIN.

d. DISK - input is from a SOurce Language Disk file (file-ID is SOLD) with corrections/inserts and change cards included from either the card reader, paper tape, or magnetic tape. The SOLD disk file is created as 80 character records, blocked five logical records per block. The TAPE option and the DISK option are not permitted on the same control card. If neither TAPE nor DISK is specified, CARD will be assumed. The NEWT option may be used with the DISK option to create a new SOLT from an old SOLD file plus changes.

e. NEWD - creates a new SOLD from cards or from an old SOLD (or SOLT) plus pertinent changes. If both the NEWT option and the NEWD option are specified, a new SOLT and a new SOLD will be created.

The NEWT option does not have to be included when operating with a SOLT, thus allowing temporary source language alterations without creating a new SOLT.

The TAPE option without the NEWT option allows a SOLT tape to be referenced and to have external source images included on the output listing and in the object program. A new SOLT will not be created. Likewise, option NEWD does not have to be included when operating with a SOLD, thus allowing temporary source language alterations without creating a new SOLD.

Columns 1-6 of the Compiler Option Control card may be left blank when compiling from cards. A sequence number is required when compiling from tape or disk when the insertion of the $ option is requested within the source input.

SOURCE DATA CARDS.

These cards follow the $ Option Control cards. The following source cards are used to create an updated version of a SOLT (or SOLD) or cause temporary changes to the SOLT (or SOLD) source language input:

a. Delete Patch Card. Punch sequence number in card columns 1-6 with the remainder of card blank.

b. Change or Addition Patch Card. Punch sequence number in card columns 1-6 and changed or added source language data in applicable card columns.

Patch card decks may reside on a labeled (users choice on file-ID) magnetic tape as 80 column, unblocked records. The deck cannot be preceded by the MCP LABEL card image, e.g., ?DATA or ?DATAB, nor be followed by the MCP's ?END end-of-file control card image. All $ CARD options are available. The systems operator must be advised of the presence of a patch deck tape so that an IL control message can be initiated at the time when the compiler requests the whereabouts of the source language input file.

The COBOL Compiler has the capability of merging inputs from two sources (punched cards or paper tape, either of which may be merged with magnetic tape) on the basis of the sequence numbers.

The COBOLL Compiler will merge inputs from two sources (punched cards or paper tape, either of which may be merged with magnetic tape or disk) also on the basis of sequence numbers.

When merging inputs, the output compilation listing will indicate all inserts and/or replacements.

All $ options may be inserted at any point within the source language input data with the exception of the TAPE and the DISK options. It is important that each $ card included in the compiler deck contains all of the desired options from point of insertion.

A card reader is not required to compile COBOLL or COBOL Programs from a SOLT magnetic tape or to compile COBOLL Programs from a SOLD disk file. Control information may be entered via the SPO or the ZIP statement when using the COBOLL or COBOL Compiler. When using the COBOL Compiler, the format is:

$$\text{CC}\ \underline{\text{COMPILE}}\ \ldots\ \text{WITH}\ \underline{\text{COBOL}}\ \begin{matrix}\underline{\text{LIBRARY}}\\ \underline{\text{SAVE}}\end{matrix}\ \underline{\text{VALUE}}\ \underline{0} = N_1\ N_2\ N_3\ N_4\ N_5\ N_6$$

where:

N_1 = 1 Tape input (SOLT)

N_2 = 0 NO LIST
1 LIST

N_3 = 0 NO NEWT
1 CREATE A NEWT

N_4 = 0 NO CODE
1 CODE

N_5 = 0 NO SUPR
1 SUPR

N_6 = 0 NO SPEC
1 SPEC

When using the COBOLL Compiler, the format is:

CC COMPILER ... WITH COBOLL $\begin{matrix}\text{LIBRARY}\\ \text{SAVE}\end{matrix}$ VALUE 0 = N_1 N_2 N_3 N_4 N_5 N_6

where:

N_1 = 1 Tape input (SOLT)
2 Disk input (SOLD)

N_2 = 0 NO LIST
1 LIST

N_3 = 0 NEWT OR NEWD
1 CREATE A NEWT
2 CREATE A NEWD
3 CREATE A NEWT AND A NEWD

N_4 = 0 NO CODE
1 CODE

N_5 = 0 NO SUPR
1 SUPR

N_6 = 0 NO SPEC
1 SPEC

NOTE

Card input is not permitted when using the SPO or ZIP statement.

LABEL EQUATION CARD.

This card may be used to change a compiler file-name in order to avoid duplication of file-names when operating in a multiprocessing environment. The format for this card is:

?COBOLL FILE (file-name) = (users choice of file-id)

or:

?COBOL FILE (file-name) = (users choice of file-id)

The Label Equation Card (or cards), if used, must immediately follow the ?COMPILE ... Control Card and precede the MCP LABEL Control Card (refer to figure 8-1).

The COBOLL and COBOL file-names which may be changed are:

a. Card file-name = DATA

b. New tape file-name = TATA

c. Old tape file-name = SOLT

The following file-names may be changed only if using the COBOLL Compiler:

a. New disk file-name = TATAD

b. Old disk file-name = SOLD

COMPILER LIMITS.

The compiler limits for the COBOLL and COBOL Compilers are as follows:

Description	COBOL (17K)	COBOLL (30K)
Data-names (excluding filler)	2100 Max.	3700 Max.
PROCEDURE DIVISION procedure-names (less one for each switch)	1750 Max.	3200 Max
Unique PICTUREs (average six characters long)	150 approx.	1000 approx.
PROCEDURE DIVISION segments	99 max.	99 max.

NOTE

> The limits above are increased with additional core. For each additional 500 characters of available core, 100 data-names or 83 PROCEDURE DIVISION procedure-names or 70 PICTUREs can be used. Request for a larger core area is made by utilizing the MCP CORE function.

Source input is in the form of card images and is limited to a maximum of 30,000 per compilation.

Work files used by the compiler will make use of multiple disk electronic units (EU's), up to a total of nine, if available. This feature results in an increased compilation speed for B 2500/B 3500 Systems configurated with multiple EU's.

Occasionally, a compile-time address error will occur due to STACK overflow. This happens, for example, when IF statements are nested too deeply or there are too many levels in a data-description for the compiler to handle with its normal STACK mechanism. When this happens, the STACK size can be changed by inserting into the ?COMPILE ... MCP Control Card, VALUE 10 = nnnnnn where nnnnnn is a 6-digit number larger than 001200 when using the COBOLL Compiler or 000800 when using the COBOL Compiler. An MCP CORE clause must also be included to increase the compiler size by an equal number of digits.

SECTION 9
READER SORTER AND LISTER

GENERAL.

This section deals with the COBOL constructs of the PROCEDURE DIVISION required to activate the READER SORTER and LISTER equipment as defined by the ASSIGN to hardware-name clause.

SPECIFIC VERB FORMATS.

The specific verb formats together with a detailed discussion of the restrictions and limitations associated with each, appear on the following pages in alphabetic sequence.

NOTES

The use of any of the following verbs requires the presence of a version of the MICR or the Combination MICR and Data Communications MCP.

See I-O-CONTROL, Section 3, for pertinent MICR input and output file handling declarations.

CONTROL 4.

The function of this verb is to cause a specified READER SORTER pocket light to become illuminated.

The construct of this clause is:

```
CONTROL 4 data-name ON file-name
```

Data-name must be declared as PICTURE 99 COMPUTATIONAL.

Data-name must contain the 2-digit pocket number which specifies the pocket light desired to be turned "ON".

Flow must be stopped and all documents pocket selected before issuing a CONTROL 4 statement.

Control is set to a NOT READY condition and must be cleared by depressing the START button on the READER SORTER.

CONTROL 6.

The function of the verb is to advance the batch counter in the READER SORTER by one.

The construct of this verb is:

```
CONTROL 6 ON file-name
```

Flow must be stopped and all documents pocket selected before issuing a CONTROL 6 statement.

OPEN

OPEN.

The function of this verb is to initiate the input processing of the READER SORTER.

The construct of this verb is:

```
                { DEMAND }
OPEN INPUT      { FLOW   }      file-name-1 [file-name-2 ...]
```

At least one of the options must be specified before a file can be read.

One statement may be used to OPEN multiple READER SORTERS in the same mode (DEMAND OR FLOW).

OPEN INPUT FLOW is required if it is desired to OPEN a file in FLOW mode.

OPEN INPUT DEMAND is assumed if DEMAND is omitted from the OPEN statement.

A CLOSE statement is required if it becomes necessary to change the mode of operation from either OPEN INPUT DEMAND or FLOW.

READ.

The function of this verb is to make available the next logical record from the READER SORTER in DEMAND or FLOW mode.

The construct of this verb is:

```
READ [FLOW] file-name procedure-name-1,  procedure-name-2
```

The record description entry (01 level) must be declared as 200 characters. Reference RECORD DESCRIPTION in the Data Division.

The READ FLOW . . . . statement must be given after an OPEN INPUT FLOW. This will start the flow of documents through the READER SORTER.

If NO-FORMAT is specified in the I-O-CONTROL procedure, the data is stored (in descending sequence) continuously. If formatting is specified (NO FORMAT is omitted), the data is stored (in descending sequence) continuously until the first transit symbol is received. Blanks are then stored until the 40th character location is reached at which point the transit symbol and remaining data is stored. Blanks are stored following the last information character read until a total of 100 characters is stored. When formatting (NO-FORMAT is omitted) is specified, automatic validity checking of the amount and transit fields is performed. Validity checking of the amount field includes checking:

a. The 1st and 12th characters stored for amount symbols.
b. The intervening 10 characters for decimal digits.

Validity checking of the transit field includes checking:

a. The 40th and 50th characters stored for transit symbols.

b. The intervening nine characters for the following: four decimal digits, hyphen (-), and four decimal digits.

Procedure-name-1 specifies the location to GO TO when the flow mode is stopped. All documents which were in motion will be processed and pocket selected before going to procedure-name-1. A READ FLOW statement has to be executed to restart the READER SORTER in a flow mode.

Procedure-name-2 specifies the location to GO TO if a batch ticket (black band) was encountered during the last document pocket selection. The record description (01 level) area has been blanked. The READER SORTER is in a stop flow mode and must be restarted with a READ FLOW statement.

It is the responsibility of the programmer to insure the presence of procedure-name-1 and procedure-name-2, otherwise, an object time error will result.

SELECT.

The function of this verb is to pocket select the last document read to the pocket specified on the SORTER-READER.

The construct of this verb is:

```
SELECT  data-name  ON  file-name  INVALID  procedure-name
```

The SELECT statement may only be executed within a USE SORTER procedure.

Data-name must be declared with a PICTURE 9(4) COMPUTATIONAL and its format is NNRV, where: NN is the pocket to be selected, R is zero, and V is either zero (if the current mode is to continue) or one (if FLOW is to be stopped).

If the SELECT for the document was too-late-to-process, the program will branch to the INVALID procedure-name. FLOW mode is stopped and the document has been sent to the reject (R) pocket. The information read from the document which caused the too-late-to-process is stored in the record description area (01 level). However, the trailing documents will not have been placed in memory, but will be routed to the reject (R) pocket.

USE.

The function of this verb is to specify procedures to handle error conditions and to process the document for pocket select when using the READER SORTER.

The construct of this verb is:

$$\underline{\text{USE}} \begin{Bmatrix}\text{ON}\\ \text{FOR}\end{Bmatrix} \underline{\text{SORTER}} \begin{Bmatrix}\underline{1}\\ \underline{2}\\ \underline{3}\\ \underline{4}\\ \underline{5}\end{Bmatrix} \begin{Bmatrix}\text{ON}\\ \text{FOR}\end{Bmatrix} \text{file-name}$$

A USE statement, when present, must immediately follow a section header in the DECLARATIVE portion of the PROCEDURE DIVISION and must be followed by a period followed by a space. The remainder of the section must consist of one or more procedural paragraphs that define the procedures to be used.

Intermediate work areas, if present, in USE ON SORTER procedures will occupy different memory areas from those used by other USE procedures, or those used by the main body of the program, thereby preventing conflict during POCKET SELECT interrupts. Note that each USE ON SORTER procedure that references an intermediate area will create a new area. Maximum use of storage will be realized when all SORTER USE procedures appear first in the DECLARATIVES. This method allows non-sorter USE procedures to share intermediate areas with the remainder of the PROCEDURE DIVISION.

Index registers are saved upon entering a USE ON SORTER procedure and will be reinstated upon exit from the procedure.

The USE statement itself is never executed rather, it defines the conditions calling for the execution of the USE procedure.

The USE. . .SORTER 1. . . option is specified to handle memory access, cannot read, unencoded, and double documents error procedures.

The USE. . .SORTER 2. . . option is specified to handle the Amount field errors.

The USE. . .SORTER 3. . . option is specified to handle the Transit field errors.

The USE. . . SORTER 4. . . option is specified for processing of a SELECT statement.

The USE. . . SORTER 5. . . option is specified for handling depression of the END-OF-FILE button on the Reader Sorter.

It is permissible to reference a non-declarative procedure when using the SELECT. . . .INVALID procedure-name statement within the USE procedures of the DECLARATIVES portion of the PROCEDURE DIVISION.

CONTROL.

The function of this verb is to space, skip, and slew specified units and tape designations on the LISTER.

The construct of this verb is:

$$\underline{\text{CONTROL}} \quad \left\{ \begin{matrix} \underline{1} \\ \underline{2} \\ \underline{3} \end{matrix} \right\} \quad \text{file-name} \quad \underline{\text{NOT-READY}} \quad \text{procedure-name}$$

The CONTROL 1. . . . option is used to perform a space operation specified by the unit and tape designations in ACTUAL KEY.

The CONTROL 2. . . . option is used to perform a skip (2 1/2 inches) operation as specified by the unit and tape designations in ACTUAL KEY.

The format of the various unit and tape designations for the CONTROL 1. . . .and CONTROL 2. . . .options are as follows:

Digit Positions			
1	2	3	4
U	T	U	T

where U equals Unit Number 1-3, U equals zero (Suppress Skip or Space), and T equals Tape Number 1-6.

For the master/slave/slave combination, a skip or space operation is performed on both the master tape of unit 1 and the tape designated by the first and second digit positions (D1-D2) of the ACTUAL KEY. If the first digit position (D1) of the ACTUAL KEY is zero, then skipping or spacing on both the master tape on unit 1 and the tape designated by the second digit position (D2) of the ACTUAL KEY is suppressed.

For the six tape/six tape combination, the second digit position of the ACTUAL KEY must equal zero, then the skipping or spacing of the master tape on the unit designated by the first digit position (D1) of the ACTUAL KEY is performed. If the first digit position (D1) of the ACTUAL KEY is equal to zero, then skipping or spacing of the master tape is suppressed.

An additional tape can be skipped or spaced as designated by the third and fourth digit positions (D3-D4) of the ACTUAL KEY.

An invalid ACTUAL KEY is when the first and third digit positions (D1 and D3) are equal to zero.

The CONTROL 3. . . .option is used to perform a slew (10 inches) operation as specified by the unit and tape designations. The ACTUAL KEY data-name clause with PICTURE 9(4) COMPUTATIONAL for slew operations is as follows:

Digit Positions

1	2	3	4
U	V	U	T

For the first and second digit positions the coding is as follows:

a. V = 0 - allow slew of master tape.

b. V = 1 - inhabit slew of master tape (refer to first note that follows).

c. U = 1 - slew all tapes, unit 1.

d. U = 2 - slew all tapes, unit 2.

e. U = 4 - slew all tapes, unit 3.

f. U = 3 - slew all tapes, units 1 and 2 (refer to second note that follows).

g. U = 5 - slew all tapes, units 1 and 3.

h. U = 6 - slew all tapes, units 2 and 3.

i. U = 7 - slew all tapes, units 1, 2, and 3.

NOTES

V takes precedence over U.

For the six tape/six tape combination, only D1 = 3 can be designated.

For the third and fourth digit positions, the coding is as follows:

a. U = unit number 1-3.

b. U = 0 - do not suppress slew.

c. T = tape number 1-6.

NOTE

D3-D4 are used only on the "18 TAPE LISTER"; otherwise D3-D4 must be zero.

ENABLE.

The function of this verb is to suspend the program until the not ready condition on the LISTER has been corrected.

The construct of this verb is:

```
ENABLE  file-name
```

File-name must have been OPENed before an ENABLE can be executed. Once the file-name has been ENABLEd, the program will be suspended until the LISTER not ready condition (not ready or end of paper) has been corrected.

WRITE.

The function of this verb is to release a logical record to be printed on the LISTER.

The construct of this verb is:

```
WRITE   record-name   NOT-READY   procedure-name
```

The ACTUAL KEY clause is required to specify the unit and tape designations. The format for the various unit and tape designations for the WRITE statements is as follows:

Digit Positions

1	2	3	4
U	T	U	T

For the first and second digit positions, the coding is as follows:

a. U = 0 - suppress print.

b. U = unit number 1-3.

c. T = tape number 1-6.

The record-name must be defined in the DATA DIVISION by means of a 01 level entry under the FD entry for the file. The 01 level entry must specify a record 44 characters in length.

For the master/slave/slave combinations, the first 22 characters of the record-name are printed on both the master tape of unit 1 and the tape designated by the first two digit positions (D1-D2) of the ACTUAL KEY. If the first digit position (D1) of the ACTUAL KEY is zero, then printing on both the master tape on unit 1 and the tape designated by the second digit position (D2) of the ACTUAL KEY is suppressed.

For the six tape/six tape combination, the second digit position (D2) of the ACTUAL KEY must be zero, then the first 22 characters are printed on the master tape of the unit designated by the first digit position (D1) of the ACTUAL KEY. If the first digit position (D1) of the ACTUAL KEY is zero, printing of the master tape is suppressed.

The second 22 characters of the record-name are printed on the tape designated by the third and fourth digit positions (D3-D4) of the ACTUAL KEY. The printing of the tape designated by the third and fourth digit positions (D3-D4) of the ACTUAL KEY, is only possible with the "18 TAPE LISTER".

If the LISTER is in a not ready condition, the WRITE. . .NOT READY procedure-name statement will be executed. If an invalid unit or tape number is specified, this can cause the LISTER to appear constantly busy. Depression of the LISTER STOP button causes the not ready condition and releases the LISTER from a busy condition.

SECTION 10
COBOL FILTER PROGRAM

GENERAL.

This section describes the usage of the Burroughs Filter Program for conversion of B 200/B 300/B 500, or B 5500, COBOL source programs to the B 2500/B 3500 USASI COBOL language.

The Filter Program is provided in two versions:

a. To be operated on a B 5500 (supplied upon request).

b. To be operated on a B 3500 (supplied on the regular ASR systems tape distribution).

Resultant SOLT tapes created by either version are acceptable as input to the B 2500/B 3500 COBOL or COBOLL (27KB version of COBOL) compilers.

A syntactically correct COBOL source program is required in order to produce correct object time results, otherwise undefined results will occur.

The Filter Program was specifically designed for B 200/B 300/B 500 and B 5500 DOD COBOL source language conversion. It will accept other than those systems specified above and has given acceptable results in several test cases, however, no guarantee of compatibility is implied or intended.

B 2500/B 3500 USASI COBOL source programs can be filtered as a precompiler symbolic language-check if so desired.

CONFIGURATION FOR EXECUTION OF FILTER.

The source program to be filtered can be read from either punched cards or magnetic tape. The input image, the output image, and any codes, error messages and/or advisory messages will appear upon the line printer. Source output data can be recorded either in punched cards or on magnetic tape. An additional magnetic tape is required if the program being filtered contains SELECT file-name RENAMING file-name statements in the ENVIRONMENT DIVISION.

The peripheral configuration required is:

B 2500/B 3500	or	B 5500
Card-Reader	or	Magnetic Tape Unit
Card-Punch	or	Magnetic Tape Unit
Printer		
Magnetic Tape Unit	if	SELECT...RENAMING...statement is present

EXECUTION CARDS.

A filter control card - FILCON - must be present following the EXECUTE and label (DATA FILTER) cards. The information can be entered in free-field format but must be in the following format order:

$$\underline{\text{FILCON}}\ \overset{\textcircled{1}}{\begin{Bmatrix}\underline{\text{CARD}}\\ \underline{\text{TAPE}}\end{Bmatrix}}\ \overset{\textcircled{2}}{\begin{Bmatrix}\underline{\text{CARD}}\\ \underline{\text{TAPE}}\\ \underline{\text{SOLT}}\\ \underline{\text{CARDEB}}\end{Bmatrix}}\ \overset{\textcircled{3}}{\begin{bmatrix}\underline{\text{NOSEQ}}\\ \underline{\text{SEQ}}\ \text{integer}\end{bmatrix}}\ \overset{\textcircled{4}}{\begin{bmatrix}\underline{\text{COBOL}}\\ \underline{\text{COBOLL}}\end{bmatrix}}\ \overset{\textcircled{5}}{\left[\underline{\text{CODE}}\right]}$$

CARD/TAPE (code ①) signifies the choice of input medium. CARD/TAPE/SOLT/CARDEB (code ②) signifies the choice of an output medium for the filtered images to be recorded on. SOLT indicates that a separate Symbolic Output Language Tape is to be created for each input source program. CARDEB indicates that the output deck will be punched in EBCDIC. CARD indicates that the output deck will be punched in BCL.

NOSEQ/SEQ integer (code ③) is a sequencing option and is coded as follows:

a. NOSEQ - output images retain the input sequence number (except when statement consolidation occurs where the last number is used).

b. SEQ - an integer from 1 through 4 digits is required to follow the entry to indicate the required sequencing interval. An interval of from 1 through 9999 is acceptable. An additional increment of 100 also occurs upon encountering an ASSIGN TO DISK phrase in the ENVIRONMENT DIVISION.

NOTE

> When neither of the above are used, output images will be sequenced by 10. An additional 100 increment occurs upon encountering an ASSIGN TO DISK phrase in the ENVIRONMENT DIVISION.

COBOL or COBOLL (code ④) provides the facility to ZIP (FILTER and compile) to the appropriate compiler. The source language program being filtered must contain a PROGRAM-ID entry if the ZIP feature is to be used. The absence of the compiler-name in the FILCON control card and/or the absence of the PROGRAM-ID entry in the source language input will disable the ZIP feature. The presence of the compiler entry indicates tape input (SOLT) and will cause an output printer listing to be created.

Code ⑤ signifies that the normal FILTER code conversion may be altered by user specifications reflecting the characters required for conversion. The CODE specification card must follow the FILCON Control Card and must contain the character pairs to be converted. The characters to be converted from, must be followed with the word BY which is followed by the characters into which the first character will be converted. For example,

```
FILCON CARD CARD NOSEQ CODE
CODE # BY = % BY ( [ BY ) @ BY " : BY " ' BY "
```

A maximum of nine single-character replacement pairs can be specified on one card.

The IBM single quotation mark (card code 5-8) will be converted to the Burroughs BCL double quotation mark (card code 0-7-8) if the conversion is specified in the CODE card.

SPECIFICATION INPUT EXAMPLE.

The order of execution can be represented as follows:

```
? EXECUTE FILTER*
? DATA FILTER
FILCON     CARD     CARD
...
program cards (may be more than one program)
...
? END
```

The above specification represents source input card images and source output images being placed to cards. Without specifying it, a printer listing will be created for input and output images, along with informational codes, messages, and error messages. A resequencing with an interval of 10 is indicated by the absence of the sequencing option.

When the TAPE option is specified for source input, a magnetic tape labeled SOLT is required. This SOLT tape must be blocked with 5 images per block (the normal output from a B 5500 or B 3500 COBOL compilation). The FILTER program will accept only the compiler created SOLT tapes, as source input, from the system it is being executed on.

The output tape will be labeled SOLT and consists of blocked (5 records per block) recorded in non-standard (BCL) mode which is suitable for input to the B 3500 COBOL Compiler.

ERROR AND ADVISORY MESSAGES.

FILCON CONTROL MISSING appearing on the console printer indicates that the FILTER control card, described above is missing. The program will automatically terminate after declaring this condition.

*The program name is FILTER for the B 3500. For the B 5500, the first card must read ? EXECUTE FILTER/COBOL.

FILCON INPUT CHOICE MISSING appearing on the console printer indicates that the FILTER input option could not be recognized. The program will automatically terminate after declaring this condition.

FILCON OUTPUT CHOICE MISSING appearing on the console printer indicates that the FILTER output option could not be recognized. The program will automatically terminate after declaring this condition.

FILCON CODE CHANGE CARD MISSING-RESTART or FILTER CODE CARD FORMAT ERROR-RESTART appearing on the console printer indicates that the CODE specification card contains an error. The program will automatically terminate after declaring this condition.

CONTROL ERROR - INSELECT - FILTER appearing on the console printer indicates that the filter is not reading input from the proper input unit. The program will automatically terminate after declaring this condition.

INPUT TAPE VERIFICATION FAILED - FILTER appearing on the console printer (SPO), and
INPUT TAPE VERIFICATION FAILED - FILTER appearing on the line printer, indicates that the tape input has failed verification as a blocked (5) tape and also has failed verification as an unblocked tape. The program will automatically terminate after declaring this condition.

ILLEGAL WORD SIZE appearing on the line printer indicates that a word size of greater than 30 characters, or that a PICTURE character-string greater than 160 characters has been introduced by the FILTER scanner. The program will continue after declaring this condition.

SOURCE PROGRAM COPY STATEMENT appearing on the line printer indicates an action that the FILTER program will not accomplish. The program will continue after declaring this condition.

PROGRAM LIBRARY REFERENCE appearing on the line printer indicates that a library reference has been made. Verification that the proper library information is available for the B 3500 system should be made. The program will continue after declaring this condition.

SELECT RENAMING LIMIT EXCEEDED appearing on the line printer indicates that more than 20 files have been selected with the RENAMING option. The additional file-names are not entered into the table and their descriptions are not added to the filtered program. The FILTER program will continue after declaring this condition.

CHANNEL ERROR appearing on the line printer indicates that the SPECIAL NAMES association specifies a line printer channel greater than 11. The program will continue after declaring this condition.

SIZE STATEMENT TO INTEGER OUT OF 6-DIGIT RANGE or
SIZE STATEMENT INTEGER OUT OF 6-DIGIT RANGE appearing on the line printer indicates that the source program size statement appears to the FILTER program to be greater than six digits. The program will continue after declaring this condition.

ILLEGAL GROUP NAME appearing on the line printer indicates that FILLER has been used as a group-name in the source image just preceding this message. The program will continue after declaring this condition.

PICTURE UNPACKER LIMIT EXCEEDED appearing on the line printer indicates that the PICTURE shown in the source image contains a representation of more than 150 characters in the PICTURE character-string. The program will continue after declaring this condition.

PICTURE PACKER LIMIT EXCEEDED appearing on the line printer indicates that the consolidation of PICTURE characters has created a PICTURE character-string larger than 150 characters. The program will continue after declaring this condition.

INCOMPLETE SPECIFICATION ERROR appearing on the line printer indicates that the preceding source image did not contain sufficient information for an elementary item. This can also be occasioned by the use of a source program COPY statement. The program will continue after declaring this condition.

OUTPUT MESSAGES AND IMAGE CODES.

The FILTER program uses identifying codes, appearing to the left of the output image on the line printer, to cause advisory messages to appear to the right of the output image on the line printer. These messages and their meaning follows:

a. A header message appears on the line printer output at the beginning of a FILTER run and consists of two lines. The first line is a copy of the FILCON control card, the second is an identification line of the type:

   FILTERED current-date USING system filter-date
   FILTER/COBOL PROGRAM

   Current-date appears in the form mm/dd/yy; system is either B 3500 or B 5500; and filter-date identifies the version of the FILTER program being used.

b. No message.
   Sequence number change only.

c. NEED MANUAL CHANGE.
   A portion of the input image is not acceptable to the B 3500 COBOL Compiler and the FILTER program is unable to accomplish a suitable correction. The programmer must review and modify this image.

d. FILTER.
   A portion of the input image has contained a construct that is not acceptable to the B 3500 COBOL Compiler, but the meaning is clear and the FILTER program has made an adjustment to the contents so that the output image is acceptable.

e. FILTER CHANGE.
   A more extensive change to the contents of the input

source image has been made. Usually, this code indicates that the entire source image has been replaced by a blank image.

f. VERIFY ENTRY/CHANGE.
The FILTER program has changed the source image in an established and usually acceptable manner; however, the programmer should verify that the modification is acceptable at this point in the program.

g. SCAN CHKFLG ERROR.
This console printer message indicates that a program switch in the scanner portion of the program is not being set properly (by the FILTER program).

h. FILTER OUTPUT SELECT ERROR.
This indicates that the output control switch has not been properly set by the FILTER program. The program will terminate after declaring this condition.

i. NEED ACCESS MODE/KEY CLAUSE.
This indicates that a file has been assigned to DISK. The control clauses appearing as part of an MD entry needs to be added in this ENVIRONMENT DIVISION entry. A sequence number increment of 100 occurs, and the program continues.

j. LEVEL ENTRY ERROR.
This indicates that a DATA DIVISION level number larger than two digits has been encountered.

k. NOTE LIMIT EXCEEDED.
This indicates that a NOTE of greater than 320 characters has appeared as part of a DATA DIVISION record description. The NOTE is truncated to 320 characters, and the program continues.

1. NOTE PARAGRAPH INDICATED.
   This indicates that the NOTE sentence is the first sentence of the PROCEDURE DIVISION paragraph, and thus the entire paragraph is a NOTE. This message will appear only when filtering a B 200/B 300/B 500 program to advise of the difference in the COBOL rules between the B 200/ B 300/B 500 and the B 3500.

REPETITIVE OPERATIONS.

More than one source program can be filtered during one execution of the FILTER program. The sensing of the IDENTIFICATION DIVISION entry of each successive program causes the FILTER program to reset itself and to begin a new conversion. If SOLT has been shown in the FILCON parameter card, separate magnetic tapes are created for each program being filtered.

A count of the number of times each output image code appears is provided, along with the count of the number of images receiving a code. These messages appear upon the printer as follows:

```
CODE 1 COUNT = nnnnn  CODE 2 COUNT = nnnnn  CODE 3 COUNT = nnnnn etc.

TOTAL IMAGES FLAGGED = NNNNN     TOTAL IMAGES FILTERED = nnnnn
```

APPENDIX A
COBOL RESERVED WORDS**

*ABOUT
ACCEPT
ACCESS
ACTUAL
ADD
*ADDRESS
ADVANCING
AFTER
ALL
ALPHABETIC
ALTER
ALTERNATE
ALTERNATING
AND
APPLY
ARE
AREA
AREAS
ASCENDING
ASSIGN
AT
ATT-8A1
AUTHOR
AUXILIARY

B-500
B-2500
B-3500
B-9350
B-9352
BACKUP
BEFORE
BEGINNING
BLANK
BLOCK
BREAK
BY
BZ

*CANCEL
*CF
*CH
CHANNEL
CHARACTERS
CLOSE
CMP
CMP-1
*CMP-3
COBOL
*CODE
*COLUMN
COMMA
COMP
*COMP-1
*COMP-3
COMPUTATIONAL
COMPUTATIONAL-1
COMPUTATIONAL-3
COMPUTE
CONFIGURATION
CONTAINS
CONTROL
CONTROL 1
CONTROL 2
CONTROL 3
CONTROL 4
CONTROL 6
*CONTROLS
CONVERSION
COPY
CORR
CORRESPONDING
CURRENCY

DATA
DATE (SPECIAL REGISTER)
DATE-COMPILED
DATE-WRITTEN
DCT-2000
*DE
DECIMAL-POINT
DECLARATIVES
DEMAND
DEPENDING
DESCENDING
*DETAIL
DIAL
DISABLE
DISC
DISCONNECT
DISK
DISPLAY
DISPLAY-UNIT
DIVIDE
DIVISION
DOWN

* These reserved words may appear in a future compiler.
** See special instructions, page 1-10.

ELSE
ENABLE
END
ENDING
END-OF-JOB
END-TEXT
END-TRANSIT
ENTER
ENVIRONMENT
EQUAL
EQUALS
ERROR
EVERY
EXAMINE
EXIT

FD
FILE
FILE-CONTROL
FILE-LIMIT
FILE-LIMITS
FILL
FILLER
*FINAL
FIRST
FLOW
*FOOTING
FOR
FORM
FROM

*GENERATE
GIVING
GO
GREATER
*GROUP

*HEADING
HIGH-VALUE
HIGH-VALUES
*HOLD

IBM-1030
IBM-1050
ID
IDENTIFICATION
IF
IGNORE
I-O
I-O-CONTROL
IN
INDEX
INDEXED
*INDICATE
*INITIATE
INPUT
INPUT-OUTPUT
INSTALLATION
INTERROGATE
INTO
INVALID
IS

JS
JUST
JUSTIFIED

KEY

LABEL
*LAST
LEADING
LEFT
LESS
LIBRARY
LIMIT
LIMITS
*LINE
*LINE-COUNTER
LINES
LISTER
LOCK
LOW-VALUE
LOW-VALUES

MEMORY
MICR
MICR-OCR
MOD
MODE
MODULES
MONITOR
MOVE
MULTIPLE
MULTIPLY

NEGATIVE
NEXT
NO
NO-DATA
NO-ERRORS
NO-FORMAT
NO-TIME-OUT
NON-STANDARD
NOT
NOT-READY

*These reserved words may appear in a future compiler.

NOTE
*NUMBER
NUMERIC

OBJECT-COMPUTER
OC
OCCURS
OCR
OF
OFF
O-I
O-L-BANKING
OMITTED
ON
OPEN
OPTIONAL
OR
OTHERWISE
OUTPUT
OVERFLOW

PAGE
*PAGE-COUNTER
PC
PERFORM
*PF
*PH
PIC
PICTURE
*PLUS
POLL
POSITION
POSITIVE
PRINTER
PRIORITY
PROCEDURE
PROCEED
*PROCESS
PROCESSING
PROCESSOR
PROGRAM-ID
PT-PUNCH
PT-READER
PUNCH
PURGE

QUOTE
QUOTES

RANDOM
*RD
READ
READER
RECORD
RECORDING
RECORDS
REDEFINES
REEL
RELEASE
REMAINDER
REMARKS
RENAMES
REPLACING
*REPORT
*REPORTING
*REPORTS
RERUN
*RESET
RESERVE
RETURN
REVERSED
REWIND
*RF
*RH
RIGHT
ROUNDED
RUN

SAME
SAVE
SAVE-FACTOR
SD
SEARCH
SECTION
SECURITY
SEEK
SEGMENT-LIMIT
SELECT
SENTENCE
SENTINEL
SEQUENTIAL
SET
SIGN
SIGNED
SIZE
SORT
SORTER
*SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
SPO
STANDARD
START-TEXT

*These. reserved words may appear in a future compiler.

*STATUS
STOP
STREAM
SUBTRACT
*SUM
SUPERVISOR
SW1
SW2
SW3
SW4
SW5
SW6
SW7
SW8
SY
SYMBOLIC
SYNC
SYNCHRONIZED

TALLY
TALLYING
TAPE
TAPE-7
TAPE-9
TC-500
*TERMINATE
THAN
THEN
THROUGH
THRU
TIME
TIMES
TO
TODAYS-DATE (SPECIAL REGISTER)
TONE
TOUCH-TONE
TRACE
TRANSLATION
TT-28
TWX
*TYPE

*UNIT
UNTIL
UP
UPON
USAGE
USASI
USE
USING

VA
VALUE
VARYING
VOICE

WAIT
WHEN
WITH
WORDS
WORK
WORKING-STORAGE
WRITE
WRITE-READ
WRITE-READ-TRANS
WRITE-TRANS-READ

ZERO
ZEROS
ZEROES
ZIP

*These reserved words may appear in a future compiler.

APPENDIX B

COBOL SYNTAX

PRE-IDENTIFICATION DIVISION.

```
[MONITOR  [DEPENDING]  file-name  ([data-name] ...   :

[{ ALL                 }]).  ]
 { paragraph-name...  }
```

IDENTIFICATION DIVISION

```
IDENTIFICATION DIVISION.
```

```
[PROGRAM-ID.  Any COBOL word.]
```

```
[AUTHOR.  Any entry.]
```

```
[INSTALLATION.  Any entry.]
```

IDENTIFICATION DIVISION (cont'd)

[DATE-WRITTEN. Any entry.]

[DATE-COMPILED. Any entry - replaced by the current date and time as maintained by the MCP.]

[SECURITY. Any entry.]

[REMARKS. Any entry. Continuation lines must be coded in Area B of the coding form.]

ENVIRONMENT DIVISION.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

ENVIRONMENT DIVISION (cont'd)

```
┌                      ┌ B-2500    ┐    ┐
│ SOURCE-COMPUTER.    <  B-3500     >  . │
└                      └ any entry ┘    ┘
```

```
┌                    ┌┌ B-2500 ┐┐
│ OBJECT COMPUTER.   ││        ││ [WITH SUPERVISOR CONTROL]
└                    └└ B-3500 ┘┘

                             ┌┌ WORDS      ┐┐
    MEMORY SIZE integer      ││ CHARACTERS ││     [TIME 60]
                             └└ MODULES    ┘┘
                                                      ┐
    [SEGMENT-LIMIT IS priority number]           .    │
                                                      ┘
```

```
┌
│ SPECIAL-NAMES.     [CURRENCY SIGN IS literal]
└

 [implementor-name IS mnemonic-name ...]
                                          ┐
 [DECIMAL-POINT IS COMMA ].               │
                                          ┘
```

```
[INPUT-OUTPUT SECTION.]
```

ENVIRONMENT DIVISION (cont'd)

Option 1:

```
[
 FILE-CONTROL.

 SELECT [OPTIONAL] file-name-1 ASSIGN TO hardware-name-1

 [ BY { FILE } ]   [ { WORK      } ]   [FOR MULTIPLE REEL]
 [    { AREA } ]   [ { PROCESSOR } ]
 [    { NN   } ]

 [ { NO TRANSLATION              } ] [ { NO BACKUP } ]   [ FORM ]
 [ { TRANSLATION NON-STANDARD    } ] [ { BACKUP    } ]   [      ]

 [ SAVE ]  [ RESERVE  { NO        }  ALTERNATE  { AREA  } ]
 [      ]  [          { integer-1 }             { AREAS } ]

 [ { FILE-LIMIT IS   } { literal-1   } { THRU    } { END         }
 [ { FILE-LIMITS ARE } { data-name-1 } { THROUGH } { literal-2   } ...
                                                   { data-name-2 }

        [ { literal-m   }  { THRU    }  { literal-n   } ] ]
        [ { data-name-m }  { THROUGH }  { data-name-n } ] ]

 [ ACCESS MODE IS { RANDOM     } ]  [ACTUAL KEY IS data-name-3]
 [                { SEQUENTIAL } ]

   [PROCESSING MODE IS SEQUENTIAL]

 [ SYMBOLIC { KEY IS   } data-name-4 [data-name-5] ... ] ... ] .
 [          { KEYS ARE }                             ]
```

ENVIRONMENT DIVISION (cont'd)

Option 2:

```
[ FILE-CONTROL.
  SELECT sort-file-name ASSIGN TO DISK. ]
```

```
[ I-O-CONTROL.
   [RERUN EVERY integer-1 RECORDS OF file-name-1] ...   [ SAME

 [ { RECORD } ] AREA FOR file-name-2 file-name-3 [file-name-4] ... ]
 [ { SORT   } ]

  [ MULTIPLE FILE TAPE "multi-file-id" CONTAINS file-name-list
   [POSITION integer-2...]...  ]

  [ APPLY  [ { MICR     } ]  [  ALTERNATING   { AREA  } ]
           [ { OCR      } ]  [                { AREAS } ]
           [ { MICR-OCR } ]

   [ WITH [NO-FORMAT] [NO-ERRORS] ]
    [END-TRANSIT] ON file-name [...]    .  ] ]
```

DATA DIVISION.

```
[FILE SECTION.]
```

Option 1:

```
FD  file-name  COPY  "library-name".
```

Option 2:

```
FD file-name-1  [ RECORDING MODE IS { STANDARD     } ]
                [                   { NON-STANDARD } ]

FILE CONTAINS integer-1 [BY integer-2] RECORDS

[ BLOCK CONTAINS [integer-3 TO]  integer-4   { RECORDS    } ]
[                                            { CHARACTERS } ]

[ RECORD CONTAINS [integer-5 TO] integer-6  CHARACTERS ]

[ LABEL  { RECORD IS   }  { OMITTED      } ]
[        { RECORDS ARE }  { STANDARD     } ]
[                         { USASI        } ]
[                         { NON-STANDARD } ]

[{ VA    }  OF  { ID             }  IS  { "literal-1"  }
[{ VALUE }      { IDENTIFICATION }      { data-name-1  }

   [SAVE-FACTOR IS literal-2]  ]

[  DATA  { RECORD IS   }  data-name-2 [data-name-3...] ]
[        { RECORDS ARE }                               ]
```

DATA DIVISION (cont'd)

Option 3:

```
SD  sort-file-name  COPY  "library-name".
```

Option 4:

```
SD  sort-file-name

 FILE CONTAINS integer-1 [BY integer-2] RECORDS

[RECORD CONTAINS [integer-3 TO integer-4 CHARACTERS] ].

[BLOCK CONTAINS [integer-5 TO] integer-6  { RECORDS    } ]
                                          { CHARACTERS }

[DATA  { RECORD IS   }  data-name-1  [data-name-2] ... ]
       { RECORDS ARE }
```

Option 1:

```
01 data-name-1 COPY "library-name".
```

DATA DIVISION (cont'd)

Option 2:

```
{ 01           }  { FILLER      }    [MOD]    [REDEFINES data-name-2]
{ level-number }  { data-name-1 }

  [ { PC      }                                           ]
  [ { PIC     }      IS      (allowable PICTURE characters) ]
  [ { PICTURE }                                           ]

  [ { BZ                }  ]  [ { OC     }
  [ { BLANK WHEN ZERO   }  ]  [ { OCCURS }

        { integer-1 TIMES                 }
        { integer-2 TO integer-3 TIMES    }

         [DEPENDING ON data-name-3]       ]

  [ { ASCENDING  }   KEY IS data-name-4   [data-name-5] ...  ] ...
  [ { DESCENDING }

  [ INDEXED BY index-name-1 [index-name-2] ...  ]
  [                                              ]

  [                 { DISPLAY          }  ]
  [                 { CMP              }  ]
  [                 { CMP-1            }  ]  [ { JS        }           ]
  [ [USAGE IS]      { COMP             }  ]  [ { JUST      }     RIGHT ]
  [                 { COMPUTATIONAL    }  ]  [ { JUSTIFIED }           ]
  [                 { COMPUTATIONAL-1  }  ]
  [                 { COMPUTATIONAL-3  }  ]
                    { INDEX            }

  [ { VA    }  IS literal-1   [ { THRU    }    literal-2 ]
  [ { VALUE }                 [ { THROUGH }              ]

    [ literal-3 ]  [ { THRU    }    literal-4 ]  ...  ]
    [           ]  [ { THROUGH }              ]       ]

  [ { SY           }    { LEFT  }  ] .
  [ { SYNC         }    { RIGHT }  ]
  [ { SYNCHRONIZED }               ]
```

DATA DIVISION (cont'd)

Option 3:

$$66\ \text{data-name-1}\ \underline{\text{RENAMES}}\ \text{data-name-2}\ \left[\begin{Bmatrix}\underline{\text{THRU}}\\ \underline{\text{THROUGH}}\end{Bmatrix}\ \text{data-name-3}\right].$$

Option 4:

$$\underline{88}\quad \text{condition-name}\quad \begin{Bmatrix}\underline{\text{VA}}\\ \underline{\text{VALUE}}\end{Bmatrix}\quad \text{IS literal-1}$$

$$[\text{literal-2...}]\ \left[\begin{Bmatrix}\underline{\text{THRU}}\\ \underline{\text{THROUGH}}\end{Bmatrix}\quad \text{literal-n...}\right].$$

$$[\underline{\text{WORKING-STORAGE SECTION}}.]$$

Same level number syntax as shown in the FILE SECTION except that 77 level numbers can be used to reflect non-contiguous data areas and if used, must precede all other level number entries.

PROCEDURE DIVISION.

```
ACCEPT  data-name  [ FROM  { SPO           } ]
                           { mnemonic-name }
```

Option 1:

```
ADD  { literal-1   }  [ { literal-2   }      ... ]
     { data-name-1 }  [ { data-name-2 }          ]

data-name-n [ROUNDED]  [ ON SIZE ERROR  any statement

[ { OTHERWISE }     statement ] ]
[ { ELSE      }               ] ]
```

Option 2:

```
ADD   { literal-1   }  [ { literal-2   }      ... ]
      { data-name-1 }  [ { data-name-2 }          ]

TO  data-name-3 [ROUNDED]   [ data-name-n [ROUNDED] ... ]

[ ON SIZE ERROR any statement   [ { OTHERWISE }      statement ] ]
[                               [ { ELSE      }                ] ]
```

PROCEDURE DIVISION (cont'd)

Option 3:

```
ADD   { literal-1   }   { literal-2   }   [ { literal-3   }  ... ]
      { data-name-1 }   { data-name-2 }   [ { data-name-3 }      ]

GIVING  data-name-n  [ROUNDED]   [ ON SIZE ERROR any

statement    [ { OTHERWISE }    statement ] ]
               { ELSE      }
```

Option 4:

```
ADD   { CORR          }   data-name-1 TO data-name-2 [ROUNDED]
      { CORRESPONDING }

[ ON SIZE ERROR any statement  [ { OTHERWISE }    statement ] ]
                                 { ELSE      }
```

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

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

```
CLOSE  file-name-1 [REEL]   [ WITH   { LOCK      } ]
                                     { PURGE     }
                                     { RELEASE   }
                                     { NO REWIND }

[file-name-2...]
```

PROCEDURE DIVISION (cont'd)

```
                                              ( data-name-2              )
COMPUTE  data-name-1  [ROUNDED] =            < numeric-literal            >
                                              ( arithmetic expression    )

[ON SIZE ERROR any statement  [ { OTHERWISE }    statement ] ]
                              [ { ELSE      }              ] ]
```

Option 1:

```
COPY  "library-name".
```

Option 2:

```
COPY  "library-name"

[REPLACING     { word-1      }   BY   { word-2      }
[              { data-name-1 }        { data-name-2 }

           [ { word-3      }   BY   { word-4      }  ... ] . ]
           [ { data-name-3 }        { data-name-4 }      ]   ]
```

```
DISPLAY   { literal-1   }     [ { literal-2   }          ... ]
          { data-name-1 }     [ { data-name-2 }              ]

   [ UPON   { SPO           } ]
   [        { mnemonic-name } ]
```

PROCEDURE DIVISION (cont'd)

Option 1:

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

[ ON SIZE ERROR any statement   [ { OTHERWISE }   statement ] ]
                                  { ELSE      }
```

Option 2:

```
DIVIDE  [MOD]   { literal-1   }  { BY   }  { literal-2   }
                { data-name-1 }  { INTO }  { data-name-2 }

   GIVING data-name-3 [ROUNDED]

  [ REMAINDER data-name-4 [ROUNDED] ]

  [ ON SIZE ERROR any statement   [ { OTHERWISE }   statement ] ]
                                    { ELSE      }
```

```
END-OF-JOB.
```

```
ENTER   { SYMBOLIC } .
        { COBOL    }
```

PROCEDURE DIVISION (cont'd)

$$\underline{\text{EXAMINE}}\ \ \text{data-name}$$

$$\left\{\begin{array}{l}
\underline{\text{TALLYING}} \left\{\begin{array}{l}\underline{\text{ALL}}\\ \underline{\text{LEADING}}\\ \underline{\text{UNTIL}}\ \underline{\text{FIRST}}\end{array}\right\} \left\{\begin{array}{l}\text{literal-1}\\ \text{data-name-1}\end{array}\right\} \left[\underline{\text{REPLACING}}\ \underline{\text{BY}} \left\{\begin{array}{l}\text{literal-2}\\ \text{data-name-2}\end{array}\right\}\right] \\ \\
\underline{\text{REPLACING}} \left\{\begin{array}{l}\underline{\text{ALL}}\\ \underline{\text{LEADING}}\\ \left[\underline{\text{UNTIL}}\right]\ \underline{\text{FIRST}}\end{array}\right\} \left\{\begin{array}{l}\text{literal-3}\\ \text{data-name-3}\end{array}\right\}\ \ \underline{\text{BY}} \left\{\begin{array}{l}\text{literal-4}\\ \text{data-name-4}\end{array}\right\}
\end{array}\right\}$$

$$\underline{\text{EXIT}}.$$

Option 1:

$$\underline{\text{FILL}}\ \text{data-name-1}\ \underline{\text{INTO}} \left\{\begin{array}{l}\text{non-numeric literal-1}\\ \text{data-name-2}\end{array}\right\}$$

$$\left[\underline{\text{PROCEED}}\ \text{TO paragraph-name}\right]$$

PROCEDURE DIVISION (cont'd)

Option 2:

```
FILL data-name-3 FROM  { non-numeric literal-2 }
                       { data-name-4           }

 [PROCEED TO paragraph-name]
```

Option 1:

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

Option 2:

```
GO TO procedure-name-1  procedure-name-2  [procedure-name-3...]

 DEPENDING ON  data-name.
```

Option 1:

```
IF  condition-1  statement-1
```

PROCEDURE DIVISION (cont'd)

Option 2:

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

Option 3:

$$\underline{\text{IF}}\quad \left\{\begin{array}{l}\text{literal-1}\\ \text{data-name-1}\\ \text{arithmetic expression-1}\end{array}\right\}\quad \text{IS}\ [\underline{\text{NOT}}]\quad \left\{\begin{array}{l} = \\ > \\ < \\ \underline{\text{EQUAL}}\ \text{TO}\\ \underline{\text{LESS}}\ \text{THAN}\\ \underline{\text{GREATER}}\ \text{THAN}\\ \underline{\text{EQUALS}}\end{array}\right\}$$

$$\left\{\begin{array}{l}\text{literal-2}\\ \text{data-name-2}\\ \text{arithmetic expression-2}\end{array}\right\}$$

Option 4:

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

PROCEDURE DIVISION (cont'd)

Option 5:

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

Option 6:

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

Option 1:

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

Option 2:

```
         { CORR          }
MOVE     { CORRESPONDING }   data-name-1 TO data-name-2
```

PROCEDURE DIVISION (cont'd)

```
MULTIPLY   { literal-1   }   BY   { literal-2   }
           { data-name-1 }        { data-name-2 }

[GIVING data-name-3]  [ROUNDED]  [ ON SIZE ERROR  any

statement    [{ OTHERWISE }    statement ]]
              { ELSE      }
```

```
label.  NOTE any comment.
```

Option 2 - Paragraph NOTE:

```
NOTE.  any comment.
```

Option 3 - Sentence NOTE:

```
NOTE any comment.
```

PROCEDURE DIVISION (cont'd)

```
OPEN

┌                        ┌ ( WITH LOCK [ACCESS] ) ┐                     ┐
│ INPUT file-name-1      │ < REVERSED           > │ [file-name-2...]    │
└                        └ ( WITH NO REWIND     ) ┘                     ┘

┌                                                                   ┐
│ OUTPUT file-name-3 [WITH NO REWIND] [file-name-4 ...]             │
└                                                                   ┘

┌ ( INPUT-OUTPUT )                   ┌                          ┐
│ < I-O          >   file-name-5     │ WITH LOCK [ACCESS]        │
└ (              )                   └                          ┘

      [file-name-6...] ┘

┌                                                    ┐
│ O-I file-name-7 [file-name-8 ...]                  │
└                                                    ┘
```

Option 1:

```
PERFORM procedure-name-1 [ { THRU    } procedure-name-2 ]
                         [ { THROUGH }                  ]
```

Option 2:

```
PERFORM procedure-name-1 [ { THRU    } procedure-name-2 ]
                         [ { THROUGH }                  ]

   { integer-1   }     TIMES
   { data-name-1 }
```

PROCEDURE DIVISION (cont'd)

Option 3:

```
PERFORM procedure-name-1 [ { THRU    } procedure-name-2 ]
                         [ { THROUGH }                  ]

   UNTIL  condition-1
```

Option 4:

```
PERFORM procedure-name-1 [ { THRU    } procedure-name-2 ]
                         [ { THROUGH }                  ]

                                 { index-name-2        }
VARYING { index-name-1 }  FROM   { data-name-2         }   BY
        { data-name-1  }         { numeric-literal-1   }

{ data-name-3         }   UNTIL  condition-1  [ AFTER { index-name-3 }
{ numeric-literal-2   }                       [       { data-name-4  }

        { index-name-4       }
FROM    { data-name-5        }   BY   { data-name-6        }
        { numeric-literal-3  }        { numeric-literal-4  }

UNTIL  condition-2 ]  [ AFTER  { index-name-5 }   FROM
                   ]  [        { data-name-7  }

{ index-name-6       }
{ data-name-8        }   BY   { data-name-9        }
{ numeric-literal-5  }        { numeric-literal-6  }

 UNTIL    condition-3 ]
```

PROCEDURE DIVISION (cont'd)

```
                            ┌┌ WITH LOCK      ┐┐┌┌ AT END      ┐
READ file-name RECORD       ││                ││││             │
                            └└ INTO data-name ┘┘└└ INVALID KEY ┘

    any imperative statement  ┌┌ OTHERWISE ┐                ┐┐
                              ││           │ any statement  ││
                              └└ ELSE      ┘                ┘┘
```

```
RELEASE record-name [FROM data-name]
```

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

   [AT END   any statement]
```

Option 1:

```
SEARCH data-name-1  ┌ VARYING  ┌ index-name-1 ┐ ┐
                    └          └ data-name-2  ┘ ┘

 ┌ AT END imperative statement-1 ┐
 └                               ┘

   WHEN condition-1  ┌ imperative statement-2 ┐
                     └ NEXT SENTENCE          ┘

 ┌ WHEN condition-2  ┌ imperative statement-3 ┐  ... ┐
 └                   └ NEXT SENTENCE          ┘      ┘
```

PROCEDURE DIVISION (cont'd)

```
SEARCH ALL data-name-3  [AT END imperative statement-4]

   WHEN condition-3  { imperative statement-5 }
                     { NEXT SENTENCE          }
```

```
SEEK  file-name  RECORD  [WITH KEY CONVERSION]
```

Option 1:

```
SET     { index-name-1 }  [ { index-name-2 }   ... ]
        { data-name-1  }  [ { data-name-2  }       ]

   TO      { index-name-3 }
           { data-name-3  }
           { literal-1    }
```

Option 2:

```
SET   index-name-4  [index-name-5 ...]

   { UP BY   }   { data-name-4 }
   { DOWN BY }   { literal-2   }
```

PROCEDURE DIVISION (cont'd)

```
SORT  file-name-1

 ⎡ ⎧ PURGE ⎫                  ⎤
 ⎢ ⎨ RUN   ⎬        ON ERROR  ⎥
 ⎣ ⎩ END   ⎭                  ⎦

   ON ⎧ DESCENDING ⎫  KEY  data-name-1 [data-name-2...]
      ⎩ ASCENDING  ⎭

 ⎡ ON ⎧ DESCENDING ⎫  KEY  data-name-3 [data-name-4...] ⎤
 ⎣    ⎩ ASCENDING  ⎭                                    ⎦

⎧ INPUT PROCEDURE IS section-name-1 ⎡⎧ THRU    ⎫ section-name-2⎤ ⎫
⎪                                   ⎣⎩ THROUGH ⎭               ⎦ ⎪
⎨                        ⎡⎧ PURGE   ⎫⎤                           ⎬
⎪ USING file-name-2      ⎢⎨ LOCK    ⎬⎥                           ⎪
⎩                        ⎣⎩ RELEASE ⎭⎦                           ⎭

⎧ OUTPUT PROCEDURE IS section-name-3 ⎡⎧ THRU    ⎫ section-name-4⎤ ⎫
⎪                                    ⎣⎩ THROUGH ⎭               ⎦ ⎪
⎨                                                                  ⎬
⎩ GIVING file-name-3 ⎡⎧ LOCK    ⎫⎤                                 ⎭
                     ⎣⎩ RELEASE ⎭⎦
```

```
STOP  ⎧ RUN     ⎫
      ⎩ literal ⎭
```

PROCEDURE DIVISION (cont'd)

Option 1:

$$\underline{\text{SUBTRACT}}\quad \left\{\begin{array}{l}\text{literal-1}\\ \text{data-name-1}\end{array}\right\}\quad \left[\left\{\begin{array}{l}\text{literal-2}\\ \text{data-name-2}\end{array}\right\}\ \ldots\right]\quad \underline{\text{FROM}}$$

$$\text{data-name-m}\ [\underline{\text{ROUNDED}}]\quad \left[\text{data-name-n}\ [\underline{\text{ROUNDED}}]\ \ldots\right]$$

$$\left[\text{ON }\underline{\text{SIZE}}\ \underline{\text{ERROR}}\text{ any statement}\quad \left[\left\{\begin{array}{l}\underline{\text{OTHERWISE}}\\ \underline{\text{ELSE}}\end{array}\right\}\text{ statement}\right]\right]$$

Option 2:

$$\underline{\text{SUBTRACT}}\quad \left\{\begin{array}{l}\text{literal-1}\\ \text{data-name-1}\end{array}\right\}\quad \left[\left\{\begin{array}{l}\text{literal-2}\\ \text{data-name-2}\end{array}\right\}\ \ldots\right]\quad \underline{\text{FROM}}$$

$$\left\{\begin{array}{l}\text{literal-m}\\ \text{data-name-m}\end{array}\right\}\quad \underline{\text{GIVING}}\quad \text{data-name-n}\ [\underline{\text{ROUNDED}}]$$

$$\left[\text{ON }\underline{\text{SIZE}}\ \underline{\text{ERROR}}\text{ any statement}\quad \left[\left\{\begin{array}{l}\underline{\text{OTHERWISE}}\\ \underline{\text{ELSE}}\end{array}\right\}\text{ statement}\right]\right]$$

Option 3:

$$\underline{\text{SUBTRACT}}\quad \left\{\begin{array}{l}\underline{\text{CORR}}\\ \underline{\text{CORRESPONDING}}\end{array}\right\}\quad \text{data-name-1 }\underline{\text{FROM}}\text{ data-name-2}$$

$$[\underline{\text{ROUNDED}}]\quad \left[\text{ON }\underline{\text{SIZE}}\ \underline{\text{ERROR}}\text{ any statement}\quad \left[\left\{\begin{array}{l}\underline{\text{OTHERWISE}}\\ \underline{\text{ELSE}}\end{array}\right\}\text{ statement}\right]\right]$$

PROCEDURE DIVISION (cont'd)

```
TRACE   { 0  }
        { 1  }
        { 2  }
        { 3  }
        { 20 }
```

```
UNLOCK  record-name
```

Option 1:

```
                                               { file-name...  }
                                               { INPUT         }
                                               { OUTPUT        }
USE AFTER STANDARD ERROR PROCEDURE ON          { INPUT-OUTPUT  }
                                               { I-O           }
                                               { O-I           }
```

Option 2:

```
USE        { AFTER  }      STANDARD       { ENDING    }
           { BEFORE }                     { BEGINNING }

[ { REEL } ]      LABEL PROCEDURE ON      { file-name... }
[ { FILE } ]                              { INPUT        }  .
                                          { OUTPUT       }
```

PROCEDURE DIVISION (cont'd)

Option 3:

```
USE   { AT END OF PAGE         }  ON file-name-1 [file-name-2...].
      { FOR KEY CONVERSION     }
```

Option 4:

```
USE  ON  STALEMATE  ON  file-name-3.
```

```
WAIT   { literal   }
       { data-name }
```

Option 1:

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

 [ { AFTER  }   ADVANCING    { { integer-1   }  LINES   } ]
 [ { BEFORE }                { { data-name-2 }          } ]
                             { [TO CHANNEL integer-2]   }

   [ TO  { ERROR     } ]
   [     { AUXILIARY } ]
```

PROCEDURE DIVISION (cont'd)

Option 2:

```
WRITE   record-name  [{ WITH LOCK      }]
                     [{ FROM data-name }]

 [ INVALID KEY any statement   [{ OTHERWISE }    statement ] ]
                               [{ ELSE      }              ]
```

```
ZIP   data-name
```

DATA COMMUNICATIONS

PROCEDURE DIVISION (cont'd)

```
ACCEPT  data-name  FROM   { literal     }
                          { data-name-1 }
```

```
CLOSE data-comm-file-name [{ WITH RELEASE  }]
                          [{ NO DISCONNECT }]
```

```
                           { BREAK      }
DISABLE file-name  [  ON   { NO-DATA    }  ]
                           { DISCONNECT }
```

```
DISPLAY     { literal-1   }  [{ literal-2   }     ... ]
            { data-name-1 }  [{ data-name-2 }         ]

UPON        { literal-3   }
            { data-name-3 }
```

```
ENABLE file-name [PROCEED TO paragraph-name]
```

DATA COMMUNICATIONS

PROCEDURE DIVISION (cont'd)

```
FILL file-name [NO-TIME-OUT] [START-TEXT] [DIAL] [END-TEXT]

                                            ( WRITE               )
                                            ( READ                )
[POLL] [VOICE] [TONE] [STREAM] WITH         { WRITE-READ          }
                                            ( WRITE-TRANS-READ    )
                                            ( WRITE-READ-TRANS    )

[PROCEED TO paragraph-name]
```

```
INTERROGATE [END-TEXT] file-name INTO data-name
```

```
READ file-name [INTO record-name] [NO-TIME-OUT] [START-TEXT]

 [STREAM] [DIAL] [AT END any statement]

 [ { OTHERWISE }      any statement ]
 [ { ELSE      }                    ]
```

Option 1:

```
WAIT
```

PROCEDURE DIVISION (cont'd)

Option 2:

```
WAIT UNTIL  { literal   }
            { data-name }
```

```
WRITE record-name-1 [FROM record-name-2] [END-TEXT]

 [START-TEXT] [STREAM] [DIAL] [VOICE] [TONE]  [AT END

 any statement]     [ { OTHERWISE }    any statement  ]
                    [ { ELSE      }                   ]
```

```
WRITE-READ record-name-1 [FROM record-name-2] [NO-TIME-OUT]

  [END-TEXT] [START-TEXT] [VOICE] [DIAL] [TONE] [POLL] [STREAM]

  [AT END any statement]
```

```
WRITE-READ-TRANS record-name-1 [FROM record-name-2]

 [NO-TIME-OUT] [DIAL]  [AT END any statement]
```

PROCEDURE DIVISION (cont'd)

```
WRITE-TRANS-READ  record-name-1 [FROM record-name-2]

 [NO-TIME-OUT] [DIAL] [AT END any statement]
```

READER SORTER AND LISTER

PROCEDURE DIVISION (cont'd)

```
CONTROL 4 data-name ON file-name
```

```
CONTROL 6 ON file-name
```

```
OPEN INPUT { DEMAND }     file-name-1 [file-name-2 ...]
           { FLOW   }
```

```
READ [FLOW] file-name procedure-name-1, procedure-name-2
```

```
SELECT  data-name  ON  file-name  INVALID  procedure-name
```

PROCEDURE DIVISION (cont'd)

$$\underline{\text{USE}} \quad \begin{Bmatrix} \text{ON} \\ \text{FOR} \end{Bmatrix} \quad \underline{\text{SORTER}} \quad \begin{Bmatrix} \underline{1} \\ \underline{2} \\ \underline{3} \\ \underline{4} \\ \underline{5} \end{Bmatrix} \quad \begin{Bmatrix} \text{ON} \\ \text{FOR} \end{Bmatrix} \quad \text{file-name}$$

$$\underline{\text{CONTROL}} \quad \begin{Bmatrix} \underline{1} \\ \underline{2} \\ \underline{3} \end{Bmatrix} \quad \text{file-name} \quad \underline{\text{NOT-READY}} \quad \text{procedure-name}$$

$$\underline{\text{ENABLE}} \quad \text{file-name}$$

$$\underline{\text{WRITE}} \quad \text{record-name} \quad \underline{\text{NOT-READY}} \quad \text{procedure-name}$$

APPENDIX C

EBCDIC,ASCII,AND BCL REFERENCE TABLES

The charts reflected in this appendix define the EBCDIC, ASCII, and BCL code specifications as implemented by the Burroughs Corporation for the B 2500/B 3500 systems.

Table C-1 and table C-2 show the USASCII X3.4 - 1963 and USASCII X3.4 - 1967 code sets respectively. The major differences in the code sets are in columns 0 and 1 (control characters) and columns 6 and 7 (lower case characters).

Table C-3 is an explanation of all characters and their functions or meanings for both the 1963 and 1967 versions of the USASCII character set. In this presentation, the first entry is the column/row notation in respect to tables C-1 and C-2. The second item is the character as it appears in the two tables, the 1963 version appearing at the top and the 1967 version on the bottom. Following the characters is the name and function of the character, if in fact it is not evident.

When reading table C-1 and C-2, and using columns and rows; the standard 7-bit character representation, with b_7 the high-order bit and b_1 the low-order bit, is shown below:

EXAMPLE:

The bit representation for the character K positioned in column 4, row 11 is:

b_7	b_6	b_5	b_4	b_3	b_2	b_1
1	0	0	1	0	1	1

The decimal equivalent of the binary number formed by bits b_7, b_6, and b_5, collectively, forms the column number, and the decimal equivalent of the binary number formed by bits b_4, b_3, b_2, and b_1, collectively, forms the row number.

Table C-4 reflects the Extended Binary Coded Decimal Interchange Code (EBCDIC) and is read exactly as tables C-1 and C-2.

Table C-1

USASCII X3.4-1963

Bits b7 b6 b5					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b4	b3	b2	b1	Row ↓ Column →	0	1	2	3	4	5	6	7
0	0	0	0	0	NULL	DC_0	ƀ	0	@	P	UNASSIGNED	UNASSIGNED
0	0	0	1	1	SOM	DC_1	!	1	A	Q		
0	0	1	0	2	EOA	DC_2	"	2	B	R		
0	0	1	1	3	EOM	DC_3	#	3	C	S		
0	1	0	0	4	EOT	DC_4	$	4	D	T		
0	1	0	1	5	WRU	ERR	%	5	E	U		
0	1	1	0	6	RU	SYNC	&	6	F	V		
0	1	1	1	7	BELL	LEM	' (APOS)	7	G	W		
1	0	0	0	8	FE_0	S0	(	8	H	X		
1	0	0	1	9	HT/SK	S1	)	9	I	Y		
1	0	1	0	10	LF	S2	*	:	J	Z		
1	0	1	1	11	VT	S3	+	;	K	[		
1	1	0	0	12	FF	S4	(COMMA) ,	<	L	\		ACK
1	1	0	0	13	CR	S5	-	=	M	]		①
1	1	1	0	14	SO	S6	.	>	N	↑		ESC
1	1	1	1	15	SI	S7	/	?	O	←		DEL

① Unassigned Control

Table C-2

USASCII X3.4-1967

Bits b7 b6 b5					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b4	b3	b2	b1	Row ↓ Column →	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,		L	\	l	¦
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	∧	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

Table C-3

1963, 1967 USASCII Characters

COLUMN/ROW	CHARACTER	
	USASCII X3.4 - 1963 USASCII X3.4 - 1967	NAME/FUNCTION
0/0	NULL NUL	Null: The all zeros character which may serve to accomplish time fill and media fill.
0/1	SOM	Start of Message: It is used in conjunction with EOA, EOM, and EOT for messages on tapes where the message is to be sent automatically.
	SOH	Start of Heading: A communication control character used at the beginning of a sequence of characters which constitute a machine-sensible address or routing information. Such a sequence is referred to as the "heading." An STX character has the effect of terminating a heading.
0/2	EOA	End of Address: This character, together with SOM, will be used to define the section of perforated tape in which the call-directing codes of the addressee are contained.
	STX	Start of Text: A communication control character which precedes a sequence of characters that are to be treated as an entity and entirely transmitted through to the ultimate destination. Such a sequence is referred to as "text." STX may be used to terminate a sequence of characters started by SOH.
0/3	EOM	End of Message: It may be used to separate individual messages which are sent in sequence on a single transmission between two stations (see SOM).
	ETX	End of Text: A communication control character used to terminate a sequence of characters started with STX and transmitted as an entity.
0/4	EOT EOT	End of Transmission: A communication control character used to indicate the conclusion of a transmission which may have contained one or more texts and any associated headings.

Table C-3 (cont)

1963, 1967 USASCII Characters

COLUMN / ROW	CHARACTER		
	USASCII X3.4 - 1963	USASCII X3.4 - 1967	NAME/FUNCTION
0/5	WRU	ENQ	Enquiry: A communication control character used in data communication systems as a request for a response from a remote station. It may be used as a "Who Are You" (WRU) to obtain identification, or may be used to obtain station status, or both.
0/6	RU		Are You: Use of this character for confirmation type of answer back has been discontinued until a more suitable arrangement can be devised.
		ACK	Acknowledge: A communication control character transmitted by a receiver as an affirmative response to a sender.
0/7	BELL	BELL	Bell: A character for use when there is a need to call for human attention. It may control alarm or attention devices.
0/8	FE	BS	Backspace: A format effector that controls the movement of the printing mechanism one print position backward on the same print line.
0/9	HT/SK	HT	Horizontal Tabulation: A format effector that controls the movement of the printing mechanism to the next in a series of predetermined positions along the print line. (Applicable also to the skip function on punched cards.)
0/10	LF	LF	Line Feed: A format effector that controls the movement of the paper one line at a time.
0/11	VT	VT	Vertical Tabulation: A format effector that controls the movement of paper to the next in a series of predetermined print lines.
0/12	FF	FF	Form Feed: A format effector that controls the movement of the printing position to the first predetermined printing line on the next form or page.

Table C-3 (cont)

1963, 1967 USASCII Characters

COLUMN / ROW	CHARACTER		
	USASCII X3.4 - 1963	USASCII X3.4 - 1967	NAME/FUNCTION
0/13	CR	CR	Carriage Return: A format effector that controls the movement of the print mechanism to the first print position on the same print line.
0/14	SO	SO	Shift Out: A control character indicating that the code combinations that follow shall be interpreted as outside of the character set of the standard code table until a Shift In character is reached.
0/15	SI	SI	Shift In: A control character indicating that the code combinations that follow shall be interpreted according to the standard code table.
1/0	DC$_0$	DLE	Data Link Escape: A communication control character that will change the meaning of a limited number of contiguously following characters. It is used exclusively to provide supplementary controls in data communication networks.
1/1 1/2 1/3 1/4	DC$_1$ DC$_2$ DC$_3$ DC$_4$		Device Controls: DC$_1$ (X-ON) turns the tape reader ON and DC$_3$ (X-OFF) turns the tape reader OFF in Models 33 and 35. DC$_2$ and DC$_4$ can be used as PUNCH-ON and PUNCH-OFF controls.
		DC$_1$ DC$_2$ DC$_3$ DC$_4$	Device Controls: Characters for the control of ancillary devices associated with data processing or telecommunication systems, more especially switching devices ON or OFF. (If a single "stop" control is required to interrupt or turn off ancillary devices, DC$_4$ is the preferred assignment.
1/5	ERR	NAK	Negative Acknowledge: A communication control character transmitted by a receiver as a negative response to the sender.
1/6	SYN	SYN	Synchronous Idle: A communication control character used by a sychronous transmission system in the absence of any other character to provide a signal from which synchronism may be achieved or retained.

Table C-3 (cont)

1963, 1967 USASCII Characters

COLUMN / ROW	CHARACTER	
	USASCII X3.4 - 1963 USASCII X3.4 - 1967	NAME/FUNCTION
1/7	LEM	Logical End of Media: Used to indicate the end of usable information, as in "End-of-Card".
	ETB	End of Transmission Block: A communication control character used to indicate the end of a block of data for communication purposes. ETB is used for blocking data where the block structure is not necessarily related to the processing format.
1/8	S0	Information Separators.
	CAN	Cancel: A control character used to indicate that the data with which it is sent is in error or is to be disregarded.
1/9	S1	Information Separators.
	EM	End of Medium: A control character associated with the sent data which may be used to identify the physical end of the medium, or the end of the used, or wanted, portion of information recorded on a medium. (The position of this character does not necessarily correspond to the physical end of the medium).
1/10	S2	Information Separator.
	SUB	Substitute: A character that may be substituted for a character which is determined to be invalid or in error.
1/11	S3	Information Separator.
	ESC	Escape: A control character intended to provide code extension (supplementary characters) in general information interchange. The Escape character itself is a prefix affecting the interpretation of a limited number of contiguously following characters.

Table C-3 (cont)

1963, 1967 USASCII Characters

	CHARACTER		
COLUMN / ROW	USASCII X3.4 - 1963	USASCII X3.4 - 1967	NAME/FUNCTION
1/12 1/13 1/14 1/15	S4 S5 S6 S7	FS GS RS US	Information Separators. File Separator, Group Separator, Record Separator, and Unit Separator: These information separators may be used within data in optional fashion, except that their hierarchical relationship shall be: FS is the most inclusive, then GS, then RS, and US is least inclusive. (The content and length of a File, Group, Record or Unit are not specified).
2/0	SP	SP	Space: A normally non-printing graphic character used to separate words. It is also a format effector which controls the movement of the printing position, one printing position forward.
2/1	!	!	Exclamation Point.
2/2	"	"	Quotation Marks (Diaeresis).
2/3	#	#	Number Sign.
2/4	$	$	Dollar Sign.
2/5	%	%	Percent.
2/6	&	&	Ampersand.
2/7	'	'	Apostrophe (Closing Single Quotation Mark; Acute Accent).
2/8	(	(	Opening Parenthesis.

Table C-3 (cont)
1963, 1967 USASCII Characters

COLUMN / ROW	CHARACTER		
	USASCII X3.4 - 1963	USASCII X3.4 - 1967	NAME/FUNCTION
2/9	)	)	Closing Parenthesis.
2/10	*	*	Asterisk.
2/11	+	+	Plus.
2/12	,	,	Comma (Cedilla).
2/13	-	-	Hyphen (Minus).
2/14	.	.	Period (Decimal Point).
2/15	/	/	Slant (Slash).
3/0	0	0	Figure Zero.
3/1	1	1	Figure One.
3/2	2	2	Figure Two.
3/3	3	3	Figure Three.
3/4	4	4	Figure Four.
3/5	5	5	Figure Five.
3/6	6	6	Figure Six.

Table C-3 (cont)

1963, 1967 USASCII Characters

COLUMN / ROW	CHARACTER		
	USASCII X3.4 - 1963	USASCII X3.4 - 1967	NAME/FUNCTION
3/7	7	7	Figure Seven:
3/8	8	8	Figure Eight,
3/9	9	9	Figure Nine,
3/10	:	:	Colon.
3/11	;	;	Semicolon,
3/12	<	<	Less Than.
3/13	=	=	Equals.
3/14	>	>	Greater Than.
3/15	?	?	Question Mark,
4/0	@	@	Commercial At,
4/1	A	A	Upper Case Letter A,
4/2	B	B	Upper Case Letter B,
4/3	C	C	Upper Case Letter C,

Table C-3 (cont)
1963, 1967 USASCII Characters

COLUMN / ROW	CHARACTER		
	USASCII X3.4 - 1963	USASCII X3.4 - 1967	NAME/FUNCTION
4/4	D	D	Upper Case Letter D,
4/5	E	E	Upper Case Letter E,
4/6	F	F	Upper Case Letter F,
4/7	G	G	Upper Case Letter G.
4/8	H	H	Upper Case Letter H.
4/9	I	I	Upper Case Letter I,
4/10	J	J	Upper Case Letter J.
4/11	K	K	Upper Case Letter K.
4/12	L	L	Upper Case Letter L.
4/13	M	M	Upper Case Letter M,
4/14	N	N	Upper Case Letter N.
4/15	O	O	Upper Case Letter O,
5/0	P	P	Upper Case Letter P,
5/1	Q	Q	Upper Case Letter Q.

Table C-3 (cont)

1963, 1967 USASCII Characters

COLUMN/ROW	CHARACTER	
	USASCII X3.4 - 1963 USASCII X3.4 - 1967	NAME/FUNCTION
5/2	R R	Upper Case Letter R,
5/3	S S	Upper Case Letter S,
5/4	T T	Upper Case Letter T
5/5	U U	Upper Case Letter U.
5/6	V V	Upper Case Letter V.
5/7	W W	Upper Case Letter W,
5/8	X X	Upper Case Letter X,
5/9	Y Y	Upper Case Letter Y,
5/10	Z Z	Upper Case Letter Z,
5/11	[[	Opening Bracket,
5/12	\\ \\	Reverse Slant,
5/13	]]	Closing Bracket.
5/14	↑ ∧	Exponentiation or Up Arrow. Circumflex,

Table C-3 (cont)

1963, 1967 USASCII Characters

<table>
<tr><th rowspan="3">COLUMN / ROW</th><th colspan="2">CHARACTER</th></tr>
<tr><th>USASCII X3.4 - 1963</th><th rowspan="2">NAME/FUNCTION</th></tr>
<tr><th>USASCII X3.4 - 1967</th></tr>
<tr><td>5/15</td><td>←</td><td>Replace by or Left Arrow.</td></tr>
<tr><td></td><td>_</td><td>Underline.</td></tr>
<tr><td>6/0</td><td></td><td>Unassigned.</td></tr>
<tr><td></td><td>`</td><td>Grave Accent (Opening Single Quotation Mark).</td></tr>
<tr><td>6/1</td><td></td><td>Unassigned.</td></tr>
<tr><td></td><td>a</td><td>Lower Case Letter a.</td></tr>
<tr><td>6/2</td><td></td><td>Unassigned.</td></tr>
<tr><td></td><td>b</td><td>Lower Case Letter b.</td></tr>
<tr><td>6/3</td><td></td><td>Unassigned.</td></tr>
<tr><td></td><td>c</td><td>Lower Case Letter c.</td></tr>
<tr><td>6/4</td><td></td><td>Unassigned.</td></tr>
<tr><td></td><td>d</td><td>Lower Case Letter d.</td></tr>
<tr><td>6/5</td><td></td><td>Unassigned.</td></tr>
<tr><td></td><td>e</td><td>Lower Case Letter e.</td></tr>
<tr><td>6/6</td><td></td><td>Unassigned.</td></tr>
<tr><td></td><td>f</td><td>Lower Case Letter f.</td></tr>
<tr><td>6/7</td><td></td><td>Unassigned.</td></tr>
<tr><td></td><td>g</td><td>Lower Case Letter g.</td></tr>
<tr><td>6/8</td><td></td><td>Unassigned.</td></tr>
<tr><td></td><td>h</td><td>Lower Case Letter h.</td></tr>
<tr><td>6/9</td><td></td><td>Unassigned.</td></tr>
<tr><td></td><td>i</td><td>Lower Case Letter i.</td></tr>
<tr><td>6/10</td><td></td><td>Unassigned.</td></tr>
<tr><td></td><td>j</td><td>Lower Case Letter j.</td></tr>
<tr><td>6/11</td><td></td><td>Unassigned.</td></tr>
<tr><td></td><td>k</td><td>Lower Case Letter k.</td></tr>
<tr><td>6/12</td><td></td><td>Unassigned.</td></tr>
<tr><td></td><td>l</td><td>Lower Case Letter l.</td></tr>
<tr><td>6/13</td><td></td><td>Unassigned.</td></tr>
<tr><td></td><td>m</td><td>Lower Case Letter m.</td></tr>
</table>

Table C-3 (cont)

1963, 1967 USASCII Characters

COLUMN / ROW	CHARACTER	
	USASCII X3.4 - 1963	NAME/FUNCTION
	USASCII X3.4 - 1967	
6/14		Unassigned.
	n	Lower Case Letter n.
6/15		Unassigned.
	o	Lower Case Letter o.
7/0		Unassigned.
	p	Lower Case Letter p.
7/1		Unassigned.
	q	Lower Case Letter q.
7/2		Unassigned.
	r	Lower Case Letter r.
7/3		Unassigned.
	s	Lower Case Letter s.
7/4		Unassigned,
	t	Lower Case Letter t.
7/5		Unassigned.
	u	Lower Case Letter u.
7/6		Unassigned.
	v	Lower Case Letter v.
7/7		Unassigned.
	w	Lower Case Letter w,
7/8		Unassigned.
	x	Lower Case Letter x,
7/9		Unassigned.
	y	Lower Case Letter y,
7/10		Unassigned.
	z	Lower Case Letter z,
7/11		Unassigned.
	{	Opening Brace.

Table C-3 (cont)

1963, 1967 USASCII Characters

COLUMN / ROW	CHARACTER	
	USASCII X3.4 - 1963 USASCII X3.4 - 1967	NAME/FUNCTION
7/12	ACK	Acknowledge: A communication control character transmitted by a receiver as an affirmative response to a sender.
	¦	Vertical line.
7/13		Unassigned Control.
	}	Closing Brace.
7/14	ESC	Mode shift character used to indicate a departure from the standard set of basic characters; e.g., used to shift from upper to lower case letters.
	~	Overline (Tilde; General Accent)
7/15	DEL DEL	Delete: This character is used primarily to "erase" or "obliterate" erroneous or unwanted characters in perforated tape. (In the strict sense, DEL is not a control character).

TABLE C-4

B 2500/B 3500

EXTENDED BINARY CODED DECIMAL INTERCHANGE CODES (EBCDIC)

CARD, 9-TRACK MAGNETIC TAPE, DISK AND MEMORY* FORMATS

BITS b8 → b7 → b6 → b5 →					0 0 0 0	0 0 0 1	0 0 1 0	0 0 1 1	0 1 0 0	0 1 0 1	0 1 1 0	0 1 1 1	1 0 0 0	1 0 0 1	1 0 1 0	1 0 1 1	1 1 0 0	1 1 0 1	1 1 1 0	1 1 1 1
b 4	b 3	b 2	b 1	COL → ↓ ROW	0	1	2	3	4	5	6	7	8	9	10 (A)	11 (B)	12 (C)	13 (D)	14 (E)	15 (F)
0	0	0	0	0	NUL 12-0-9-8-1	DLE 12-11-9-8-1	11-0-9-8-1	12-11-0-9-8-1	SP BLANK	& 12	- 11	12-11-0	12-0-8-1	12-11-8-1	11-0-8-1	12-11-0-8-1	PZ (+) 12-0	MZ (!) 11-0	0-8-2	0 0
0	0	0	1	1	SOH 12-9-1	DC1 11-9-1	0-9-1	9-1	12-0-9-1	12-11-9-1	/ 0-1	12-11-0-9-1	a 12-0-1	j 12-11-1	11-0-1	12-11-0-1	A 12-1	J 11-1	11-0-9-1	1 1
0	0	1	0	2	STX 12-9-2	DC2 11-9-2	0-9-2	SYN 9-2	12-0-9-2	12-11-9-2	11-0-9-2	12-11-0-9-2	b 12-0-2	k 12-11-2	s 11-0-2	12-11-0-2	B 12-2	K 11-2	S 0-2	2 2
0	0	1	1	3	ETX 12-9-3	DC3 11-9-3	0-9-3	9-3	12-0-9-3	12-11-9-3	11-0-9-3	12-11-0-9-3	c 12-0-3	l 12-11-3	t 11-0-3	12-11-0-3	C 12-3	L 11-3	T 0-3	3 3
0	1	0	0	4	12-9-4	11-9-4	0-9-4	9-4	12-0-9-4	12-11-9-4	11-0-9-4	12-11-0-9-4	d 12-0-4	m 12-11-4	u 11-0-4	12-11-0-4	D 12-4	M 11-4	U 0-4	4 4
0	1	0	1	5	HT 12-9-5	NL 11-9-5	LF 0-9-5	9-5	12-0-9-5	12-11-9-5	11-0-9-5	12-11-0-9-5	e 12-0-5	n 12-11-5	v 11-0-5	12-11-0-5	E 12-5	N 11-5	V 0-5	5 5
0	1	1	0	6	12-9-6	BS 11-9-6	ETB 0-9-6	9-6	12-0-9-6	12-11-9-6	11-0-9-6	12-11-0-9-6	f 12-0-6	o 12-11-6	w 11-0-6	12-11-0-6	F 12-6	O 11-6	W 0-6	6 6
0	1	1	1	7	DEL 12-9-7	11-9-7	ESC 0-9-7	EOT 9-7	12-0-9-7	12-11-9-7	11-0-9-7	12-11-0-9-7	g 12-0-7	p 12-11-7	x 11-0-7	12-11-0-7	G 12-7	P 11-7	X 0-7	7 7
1	0	0	0	8	12-9-8	CAN 11-9-8	0-9-8	9-8	12-0-9-8	12-11-9-8	11-0-9-8	12-11-0-9-8	h 12-0-8	q 12-11-8	y 11-0-8	12-11-0-8	H 12-8	Q 11-8	Y 0-8	8 8
1	0	0	1	9	12-9-8-1	EM 11-9-8-1	0-9-8-1	9-8-1	12-8-1	11-8-1	0-8-1	8-1	i 12-0-9	r 12-11-9	z 11-0-9	12-11-0-9	I 12-9	R 11-9	Z 0-9	9 9
1	0	1	0	10 (A)	12-9-8-2	11-9-8-2	0-9-8-2	9-8-2	[12-8-2	] 11-8-2	12-11	: 8-2	12-0-8-2	12-11-8-2	11-0-8-2	12-11-0-8-2	12-0-9-8-2	12-11-9-8-2	11-0-9-8-2	12-11-0-9-8-2
1	0	1	1	11 (B)	VT 12-9-8-3	11-9-8-3	0-9-8-3	9-8-3	. 12-8-3	$ 11-8-3	, 0-8-3	# 8-3	12-0-8-3	12-11-8-3	11-0-8-3	12-11-0-8-3	12-0-9-8-3	12-11-9-8-3	11-0-9-8-3	12-11-0-9-8-3
1	1	0	0	12 (C)	FF 12-9-8-4	FS 11-9-8-4	0-9-8-4	DC4 9-8-4	< 12-8-4	* 11-8-4	% 0-8-4	@ 8-4	12-0-8-4	12-11-8-4	11-0-8-4	12-11-0-8-4	12-0-9-8-4	12-11-9-8-4	11-0-9-8-4	12-11-0-9-8-4
1	1	0	1	13 (D)	CR 12-9-8-5	GS 11-9-8-5	ENQ 0-9-8-5	NAK 9-8-5	(12-8-5	) 11-8-5	(US) 0-8-5	' 8-5	12-0-8-5	12-11-8-5	11-0-8-5	12-11-0-8-5	12-0-9-8-5	12-11-9-8-5	11-0-9-8-5	12-11-0-9-8-5
1	1	1	0	14 (E)	SO 12-9-8-6	RS 11-9-8-6	ACK 0-9-8-6	9-8-6	+ 12-8-6	; 11-8-6	> 0-8-6	= 8-6	12-0-8-6	12-11-8-6	11-0-8-6	12-11-0-8-6	12-0-9-8-6	12-11-9-8-6	11-0-9-8-6	12-11-0-9-8-6
1	1	1	1	15 (F)	SI 12-9-8-7	US 11-9-8-7	BEL 0-9-8-7	SUB 9-8-7	| 12-8-7	¬ 11-8-7	? 0-8-7	" 8-7	12-0-8-7	12-11-8-7	11-0-8-7	12-11-0-8-7	DELIMETER 12-0-9-8-7	12-11-9-8-7	11-0-9-8-7	12-11-0-9-8-7

$b_8 b_7 b_6 b_5 / b_4 b_3 b_2 b_1$

*INTERNAL COLLATING SEQUENCE = 0 0 0 0/0 0 0 0 TO 1111/1111

TABLE C-4 (cont'd)

CONTROL AND SPECIAL CODES

NUL NULL - The all-zeros character which may serve to accomplish time fill and media fill.

SOH START OF HEADING - A communication control character used at the beginning of a sequence of characters which constitutes a machine-sensible address or routing information. Such a sequence is referred to as the "heading." An STX character has the effect of terminating a heading.

STX START OF TEXT - A communication control character which precedes a sequence of characters that is to be treated as an entity and entirely transmitted through to the ultimate destination. Such a sequence is referred to as "TEXT." STX may be used to terminate a sequence of characters started by SOH.

ETX END OF TEXT - A communication control character used to terminate a sequence of characters started with STX and transmitted as an entity.

HT HORIZONTAL TABULATION - A format effector which controls the movement of the printing position to the next in a series of predetermined positions along the printing line. (Applicable also to display devices and the SKIP function on punched cards.)

DEL DELETE - This character is used primarily to "ERASE" or "OBLITERATE" erroneous or unwanted characters in perforated tape. (In the strict sense, DEL is not a control character.)

VT VERTICAL TABULATION - A format effector which controls the movement of the printing position to the next in a series of predetermined printing lines. (Applicable also to display devices.)

FF FORM FEED - A format effector which controls the movement of the printing position to the first predetermined printing line on the next form or page. (Applicable also to display devices.)

CR CARRIAGE RETURN - A format effector which controls the movement of the printing position to the first printing position on the same printing line. (Applicable also to display devices.)

SO SHIFT OUT - A control character indicating that the code combinations which follow shall be interpreted as outside of the character set of the Standard Code Table until a shift in character is reached.

SI SHIFT IN - A control character indicating that the code combinations which follow shall be interpreted according to the Standard Code Table.

DLE DATA LINK ESCAPE - A communication control character which will change the meaning of a limited number of contiguously following characters. It is used exclusively to provide supplementary controls in data communication networks.

DC1
DC2
DC3
DC4 DEVICE CONTROLS - Characters for the control of ancillary devices associated with Data Processing or Telecommunication Systems, more especially switching devices "ON" or "OFF." (If a single "STOP" control is required to interrupt or turn off ancillary devices, DC4 is the preferred assignment.)

NL NEW LINE - A format effector which causes both Carriage Return and Line Feed.

BS BACKSPACE - A format effector which controls the movement of the printing position one printing space backward on the same printing line. (Applicable also to display devices.)

CAN CANCEL - A control character used to indicate that the data with which it is sent is in error or is to be disregarded.

EM END OF MEDIUM - A control character associated with the sent data which may be used to identify the physical end of the medium, or the end of the used, or wanted, portion of information recorded on a medium. (The position of this character does not necessarily correspond to the physical end of the medium.)

FS FILE SEPARATOR
GS GROUP SEPARATOR
RS RECORD SEPARATOR
US UNIT SEPARATOR } These information separators may be used within data in optional fashion, except that their hierarchical relationship shall be: FS is the most inclusive, then GS, then RS, and US is least inclusive. (The content and length of a File, Group, Record, or Unit are not specified.)

LF LINE FEED - A format effector which controls the movement of the printing position to the next printing line. (Applicable also to display devices.)

ETB END OF TRANSMISSION BLOCK - A communication control character used to indicate the end of a block of data for communication purposes. ETB is used for blocking data where the block structure is not necessarily related to the processing format.

ESC ESCAPE - A control character intended to provide code extension (supplementary characters) in General Information Interchange. The escape character itself is a prefix affecting the interpretation of a limited number of contiguously following characters.

ENQ ENQUIRY - A communication control character used in Data Communication Systems as a request for a response from a Remote Station. It may be used as a "WHO YOU ARE" (WRU) to obtain identification, or may be used to obtain Station Status, or both.

ACK ACKNOWLEDGE - A communication control character transmitted by a receiver as an affirmative response to a sender.

BEL BELL - A character for use when there is a need to call for human attention. It may control alarm or attention devices.

SYN SYNCHRONOUS IDLE - A communication control character used by a synchronous transmission system in the absence of any other character to provide a signal from which synchronism may be achieved or retained.

EOT END OF TRANSMISSION - A communication control character used to indicate the conclusion of a transmission, which may have contained one or more texts and any associated headings.

NAK NEGATIVE ACKNOWLEDGE - A communication control character transmitted by a receiver as a negative response to the sender.

SUB SUBSTITUTE - A character that may be substituted for a character which is determined to be invalid or in error.

SP SPACE - A normally non-printing graphic character used to separate words. It is also a format effector which controls the movement of the printing position, one printing position forward. (Applicable also to display devices.)

OTHER CODES

PZ PLUS ZERO
MZ MINUS ZERO } Code 01001110, is never obtained from a BCL plus sign; plus zero (PZ) code 1100 0000 prints as a plus sign. Minus zero (MZ) prints as an !. Choice of graphic for PZ and MZ may vary from system to system. The choice of styling (! or |) for graphic code 0100 1111 may also vary from system to system.

Table C-5 reflects the internal EBCDIC structure in its sequential code arrangement for B 2500/B 3500 Systems, plus the USASCII and BCL magnetic tape coding structures.

The two methods of creating the two-character codes representing these structures are broken down as follows:

a. 8-bit (byte) character code:

Converted	Decimal Equivalent		Binary			Decimal Equivalent	Converted
	0		0000	0000		0	
	1		0001	0001		1	
	2		0010	0010		2	
	3		0011	0011		3	
	9		1001	1001		9	
A	10		1010	1010		10	A
B	11		1011	1011		11	B
C	12	Undigit	1100	1100	Undigit	12	C
D	13	Character	1101	1101	Character	13	D
E	14		1110	1110		14	E
F	15		1111	1111		15	F

Example:

If a memory dump reflects A3, the internal code would be 1010 0011. The highest sequential code is FF and the internal code is 1111 1111.

b. 6-bit (byte) character code:

Decimal	Binary		Decimal	Converted
0	00	0000	0	
1	01	0100	4	
2	10	1001	9	
3	11	1010	10	A
0	00	1100	12	C
2	10	1111	15	F

c. The graphics appearing in the BCL column reflect differences between EBCDIC and BCL and those pertinent ones left blank reflect that the graphics of each are the same. This method holds true for the card code column of the BCL description also.

NOTE

There are only 64 unique 6-bit BCL tape codes. Therefore, where no BCL tape code is indicated in the table, there will be no BCL graphic character.

Table C-5

Internal EBCDIC Characters

EBCDIC			USASCII	BCL		
8-Bit Internal Code	Graphic	Card Code	8-Bit Internal Code	6-Bit Tape Code	Graphic	Card Code
00	NULL	12-0-9-8-1	80			
01	SOH	12-9-1	81			
02	STX	12-9-2	82			
03	ETX	12-9-3	83			
04		12-9-4	84			
05	HT	12-9-5	85			
06		12-9-6	86			
07	DEL	12-9-7	87			
08		12-9-8	88			
09		12-9-8-1	89			
0A		12-9-8-2	8A			
0B	VT	12-9-8-3	8B			
0C	FF	12-9-8-4	8C			
0D	CR	12-9-8-5	8D			
0E	SO	12-9-8-6	8E			
0F	SI	12-9-8-7	8F			

Table C-5 (cont)

Internal EBCDIC Characters

EBCDIC			USASCII	BCL		
8-Bit Internal Code	Graphic	Card Code	8-Bit Internal Code	6-Bit Tape Code	Graphic	Card Code
10	DLE	12-11-9-8-1	90			
11	DC1	11-9-1	91			
12	DC2	11-9-2	92			
13	DC3	11-9-3	93			
14		11-9-4	94			
15	NL	11-9-5	95			
16	BS	11-9-6	96			
17		11-9-7	97			
18	CAN	11-9-8	98			
19	EM	11-9-8-1	99			
1A		11-9-8-2	9A			
1B		11-9-8-3	9B			
1C	FS	11-9-8-4	9C			
1D	GS	11-9-8-5	9D			
1E	RS	11-9-8-6	9E			
1F	US	11-9-8-7	9F			
20		11-0-9-8-1				
21		0-9-1				
22		0-9-2				
23		0-9-3				
24		0-9-4				
25	LF	0-9-5				
26	ETB	0-9-6				
27	ESC	0-9-7				
28		0-9-8				
29		0-9-8-1				
2A		0-9-8-2				
2B		0-9-8-3				
2C		0-9-8-4				
2D	ENQ	0-9-8-5				
2E	ACK	0-9-8-6				
2F	BEL	0-9-8-7				
30		12-11-0-9-8-1				
31		9-1				
32	SYN	9-2				
33		9-3				
34		9-4				
35		9-5				
36		9-6				
37	EOT	9-7				
38		9-8				
39		9-8-1				
3A		9-8-2				
3B		9-8-3				
3C	DC4	9-8-4				

Table C-5 (cont)

Internal EBCDIC Characters

EBCDIC			USASCII	BCL		
8-Bit Internal Code	Graphic	Card Code	8-Bit Internal Code	6-Bit Tape Code	Graphic	Card Code
3D	NAK	9-8-5				
3E		9-8-6				
3F	SUB	9-8-7				
40	SPACE		A0	10		
41		12-0-9-1				
42		12-0-9-2				
43		12-0-9-3				
44		12-0-9-4				
45		12-0-9-5				
46		12-0-9-6				
47		12-0-9-7				
48		12-0-9-8				
49		12-8-1				
4A	[	12-8-2	DB	3C		12-8-4
4B	.	12-8-3	AE	3B		
4C	<	12-8-4	BC	3E		12-8-6
4D	(	12-8-5	A8	3D		
4E	+	12-8-6	AB	3A		12-0
4F	!	12-8-7	DE	3F	←	
50	&	12	A6	30		
51		12-11-9-1				
52		12-11-9-2				
53		12-11-9-3				
54		12-11-9-4				
55		12-11-9-5				
56		12-11-9-6				
57		12-11-9-7				
58		12-11-9-8				
59		11-8-1				
5A	]	11-8-2	DD	1E		0-8-6
5B	$	11-8-3	A4	2B		
5C	*	11-8-4	AA	2C		
5D	)	11-8-5	A9	2D		
5E	;	11-8-6	BB	2E		
5F	¬	11-8-7	DC	2F	≤	
60	-	11	AD	20		
61	/	0-1		11		
62		11-0-9-2				
63		11-0-9-3				
64		11-0-9-4				
65		11-0-9-5				
66		11-0-9-6				
67		11-0-9-7				
68		11-0-9-8				
69		0-8-1				

Table C-5 (cont)

Internal EBCDIC Characters

EBCDIC			USASCII	BCL		
8-Bit Internal Code	Graphic	Card Code	8-Bit Internal Code	6-Bit Tape Code	Graphic	Card Code
6A		12-11				
6B	,	0-8-3	AC	1B		
6C	%	0-8-4	A5	1C		
6D	Underscore	0-8-5	DF	1A	≠	0-8-2
6E	>	0-8-6	BE	0E		8-6
6F	?	0-8-7	BF	00		
70		12-11-0				
71		12-11-0-9-1				
72		12-11-0-9-2				
73		12-11-0-9-3				
74		12-11-0-9-4				
75		12-11-0-9-5				
76		12-11-0-9-6				
77		12-11-0-9-7				
78		12-11-0-9-8				
79		8-1				
7A	:	8-2	BA	0N		8-5
7B	#	8-3	A3	0B		
7C	@	8-4	C0	0C		
7D	'	8-5	A7	0F	≥	8-7
7E	=	8-6	BD	1D		0-8-5
7F	"	8-7	A2	1F		0-8-7
80		12-0-8-1				
81	a	12-0-1				
82	b	12-0-2				
83	c	12-0-3				
84	d	12-0-4				
85	e	12-0-5				
86	f	12-0-6				
87	g	12-0-7				
88	h	12-0-8				
89	i	12-0-9				
8A		12-0-8-2				
8B		12-0-8-3				
8C		12-0-8-4				
8D		12-0-8-5				
8E		12-0-8-6				
8F		12-0-8-7				
90		12-11-8-1				
91	j	12-11-1				
92	k	12-11-2				
93	l	12-11-3				
94	m	12-11-4				
95	n	12-11-5				
96	o	12-11-6				

Table C-5 (cont)

Internal EBCDIC Characters

EBCDIC			USASCII	BCL		
8-Bit Internal Code	Graphic	Card Code	8-Bit Internal Code	6-Bit Tape Code	Graphic	Card Code
97	p	12-11-7				
98	q	12-11-8				
99	r	12-11-9				
9A		12-11-8-2				
9B		12-11-8-3				
9C		12-11-8-4				
9D		12-11-8-5				
9E		12-11-8-6				
9F		12-11-8-7				
A0		11-0-8-1				
A1		11-0-1				
A2	s	11-0-2				
A3	t	11-0-3				
A4	u	11-0-4				
A5	v	11-0-5				
A6	w	11-0-6				
A7	x	11-0-7				
A8	y	11-0-8				
A9	z	11-0-9				
AA		11-0-8-2				
AB		11-0-8-3				
AC		11-0-8-4				
AD		11-0-8-5				
AE		11-0-8-6				
AF		11-0-8-7				
B0		12-11-0-8-1				
B1		12-11-0-1				
B2		12-11-0-2				
B3		12-11-0-3				
B4		12-11-0-4				
B5		12-11-0-5				
B6		12-11-0-6				
B7		12-11-0-7				
B8		12-11-0-8				
B9		12-11-0-9				
BA		12-11-0-8-2				
BB		12-11-0-8-3				
BC		12-11-0-8-4				
BD		12-11-0-8-5				
BE		12-11-0-8-6				
BF		12-11-0-8-7				
C0	(+)PZ	12-0				
C1	A	12-1	C1	31		
C2	B	12-2	C2	32		
C3	C	12-3	C3	33		

Table C-5 (cont)

Internal EBCDIC Characters

EBCDIC			USASCII	BCL		
8-Bit Internal Code	Graphic	Card Code	8-Bit Internal Code	6-Bit Tape Code	Graphic	Card Code
C4	D	12-4	C4	34		
C5	E	12-5	C5	35		
C6	F	12-6	C6	36		
C7	G	12-7	C7	37		
C8	H	12-8	C8	38		
C9	I	12-9	C9	39		
CA		12-0-9-8-2				
CB		12-0-9-8-3				
CC		12-0-9-8-4				
CD		12-0-9-8-5				
CE		12-0-9-8-6				
CF		12-0-9-8-7				
D0	(!)MZ	11-0	A1	2A	x	11-0
D1	J	11-1	CA	21		
D2	K	11-2	CB	22		
D3	L	11-3	CC	23		
D4	M	11-4	CD	24		
D5	N	11-5	CE	25		
D6	O	11-6	CF	26		
D7	P	11-7	D0	27		
D8	Q	11-8	D1	28		
D9	R	11-9	D2	29		
DA		12-11-9-8-2				
DB		12-11-9-8-3				
DC		12-11-9-8-4				
DD		12-11-9-8-5				
DE		12-11-9-8-6				
DF		12-11-9-8-7				
E0		0-8-2				
E1		11-0-9-1				
E2	S	0-2	D3	12		
E3	T	0-3	D4	13		
E4	U	0-4	D5	14		
E5	V	0-5	D6	15		
E6	W	0-6	D7	16		
E7	X	0-7	D8	17		
E8	Y	0-8	D9	18		
E9	Z	0-9	DA	19		
EA		11-0-9-8-2				
EB		11-0-9-8-3				
EC		11-0-9-8-4				
ED		11-0-9-8-5				
EE		11-0-9-8-6				
EF		11-0-9-8-7				

Table C-5 (cont)

Internal EBCDIC Characters

EBCDIC			USASCII	BCL		
8-Bit Internal Code	Graphic	Card Code	8-Bit Internal Code	6-Bit Tape Code	Graphic	Card Code
F0	0	0	B0	0A		
F1	1	1	B1	01		
F2	2	2	B2	02		
F3	3	3	B3	03		
F4	4	4	B4	04		
F5	5	5	B5	05		
F6	6	6	B6	06		
F7	7	7	B7	07		
F8	8	8	B8	08		
F9	9	9	B9	09		
FA		12-11-0-9-8-2				
FB		12-11-0-9-8-3				
FC		12-11-0-9-8-4				
FD		12-11-0-9-8-5				
FE		12-11-0-9-8-6				
FF		12-11-0-9-8-7				

APPENDIX D

WARNING AND DIAGNOSTIC ERROR MESSAGES

Output listings generated from COBOL compilations will highlight warning and syntactical errors by placing a number of the left hand side of the listing which will relate to a descriptive warning or error message printed on the right hand side. An entry will appear at the end of the compilation listing which states LAST ERROR AT SEQUENCE NUMBER nnnnnn to indicate the last sequence number where an error occurred. Reference to the last sequence error will identify the preceding line number where an error has occurred, and so on, until finally the FIRST error line is located. Each message will cause a row of X's to be printed which act as a cursor to identify the location of the error in the statement line. For example:

```
                    ...
                    ...
                    ...
                    009155   FD  TAPEIN
      FIRST ERROR 170   XXX                   FILE NOT SELECTED
                    ...
                    ...
                    ...
                    115110   OPEN INPUT PRINT-OUT
     009155 ERROR 390   XXXXXXXXXXX           UNIDENTIFIED WORD
                    ...
                    ...
                    ...
     OBJECT CODE NOT GENERATED, LAST ERROR AT SEQUENCE NUMBER 115110
```

Error messages in the PROCEDURE DIVISION usually immediately follow the line in error and those pertaining to other divisions usually precede the line in error.

The following error messages will be printed at the end of the compilation listing and are not affiliated with an error number:

Error Message	Description
CODE FILE ROW SIZE WAS EXCEEDED	Any one program segment is limited to 100,000 digits. Use sections and priority numbers to divide into smaller segments. Programs cannot be executed.

Error Message	Description
DATA DIVISION AND CONSTANTS EXCEEDS 100,000	DATA DIVISION and constant pool address are restricted to first 100,000 digits of the program. Program may be executed but references to data above 100,000 (if any) will be wrong. Data size can be shortened by reducing the number of items being MONITORed, the data declarations in the DATA DIVISION, or the size of literals in the PROCEDURE DIVISION.
PROGRAM ADDRESS REGISTER EXCEEDS 300,000	No program may require more than 300,000 digits of core. Reduce size by segmentation.

The warning messages in table D-1 are in sequence by warning message number.

Table D-1

Compiler Warning Messages

Warning Message Number	Message	Description
10	Filler Added	Indicates that from 1 to 3 digits have been added in order to start the next group level at mod 2 or a record at mod 4.
20	Receiving Field Truncation	The sending field is longer than the receiving field and some data will be lost.
30	Sequence Error	The value in Card Column 1 thru 6 of the source program statement is not greater than the preceding value.
40	Block Not Multiple of Record Size	Record size will not divide evenly into the block size.

Table D-1 (cont)

Compiler Warning Messages

Warning Message Number	Message	Description
50	Possible CMP Group Usage Error	Group computational receiving fields are treated as if they are alphanumerically declared and will not convert data from one form to another (CMP or DISPLAY).
60	Seek/Fill Needs ALT. Area	If a seek or fill statement is used in a program then at least one alternate area must be reserved.
70	Records/Area Made Multiple Of Block Size	If the file contains factor is not an even multiple of the blocking factor the compiler will adjust the row size upware until it becomes an even multiple of the block size.
120	Perform Exit Occurs Before Enter	If B thru A is performed rather than A thru B the exit for the perform will occur in program sequence prior to the enter.

The syntactical error messages in table D-2 are in sequence by error number. The description adjacent to the error number defines the cause of the error. There is a group and error message number. The group error number identifies the general error grouping with the descriptive title that will be printed on the listing. The error message number and the adjacent description defines the cause(s) of the error. The column labeled Verb contains the compiler verb which pertains to that specific error message number. The verb and some of the procedure-names should be helpful in determining the cause of the error. Those error messages without a verb or procedure entry are omitted due to being of no value to the programmer.

Table D-2
Compiler Error Messages

Error Message Number	Description	Verb
	150 Group - DUPLICATED WORD	
150	Name was previously used and may not be used again since it cannot be qualified in this case.	
151	File was previously selected.	FILE-CONTROL
152	Nested COPY statements not allowed.	
	160 Group - EXCESS OPERANDS	
160	Too many operands. Usually a result field series where series not allowed.	
	170 Group - FILE NOT SELECTED	
170	Missing file-name.	
170	Missing file-name, INPUT or OUTPUT.	USE
170	Construct requires at least one file.	
170	File has not been selected.	FILE SECTION
170	Missing file-name.	
171	File not selected in MONITER statement.	
	180 Group - ILLEGAL LITERAL	
180	Literal too large.	
180	Literal operand greater than 98 digits in length.	

Table D-2 (cont)
Compiler Error Messages

Error Message Number	Description	Verb
180	Left quote mark missing on continuation card.	
180	Literal may not follow GIVING.	
180	Illegal literal.	
181	Right quote mark missing on continuation card.	
182	Literal greater than 160 characters in length.	
183	Literal has scale greater than 99 digits.	
184	Literal has size of zero (i.e., "").	
185	Misspelled literal (e.g., 123.AA).	
186	Illegal literal in SEGMENT-LIMIT or currency clause.	
187	Illegal priority number.	
	<u>190 Group - ILLEGAL OPERAND</u>	
190	ALL has been used with data-name.	MOVE
190	Data-name has not been declared in the DATA DIVISION.	MOVE
190	CORRESPONDING operand error.	
190	Class rules between sending and receiving items has been violated.	MOVE

Table D-2 (cont)

Compiler Error Messages

Error Message Number	Description	Verb
190	Literal following ALL has scale part.	MOVE
190	Item must be defined as elementary numeric.	PERFORM
190	Literal (or data-name used as such) size not one.	EXAMINE
190	Item being examined not DISPLAY USAGE.	EXAMINE
190	Item being examined not a data-name.	EXAMINE
190	Data-name (used as a literal) not DISPLAY USAGE.	EXAMINE
190	Illegal operand.	
190	Data-name (used as a literal) is subscripted.	EXAMINE
190	Operand greater than 98 digits in length.	
190	Operand not a literal (or data-name used as such).	EXAMINE
190	Illegal operand in DISPLAY or ACCEPT. Must be a group or elementary item or a special register.	ACCEPT/ DISPLAY
190	Literal class not consistent with examined data-name.	EXAMINE
190	Illegal literal. Must be DISPLAY.	

Table D-2 (cont)

Compiler Error Messages

Error Message Number	Description	Verb
190	Data-name following INTERROGATE must be a 16-digit computational elementary item or an 8-character group item.	DATA-COMM
190	Must be a data-name declared as PC 9(5), CMP, or a literal less than 100,000.	DATA-COMM
190	ALTER operand is not a switch.	ALTER
190	Cannot GO TO label in DECLARATIVES.	GO TO
190	GO TO operand not a label.	GO TO
191	Missing file-name.	
191	Missing operand.	
191	Missing integer.	
192	Scope error.	RENAME
193	Number of records missing in RERUN clause.	I-O-CONTROL
194	Illegal BLOCK contains VALUE.	FILE-CONTROL
195	Exponent must be an integer. If literal, the size must be five or less. If data-name, it must be unsigned and less than five in value.	
196	Illegal SAVE-FACTOR value.	FILE-CONTROL
197	Illegal LABEL RECORD declaration.	FILE-CONTROL

Table D-2 (cont)

Compiler Error Messages

Error Message Number	Description	Verb
198	Illegal RECORDING MODE declaration.	FILE-CONTROL
199	File not declared.	I-O-CONTROL
	200 Group - ILLEGAL PICTURE CATAGORY	
201	Repeat part of PICTURE in error.	PICTURE
202	Float characters not valid.	PICTURE
203	Invalid sequence of PICTURE characters.	PICTURE
204	P PICTURE character(s) in error.	PICTURE
205	Decimal point in error.	PICTURE
206	Sign error.	PICTURE
207	PICTURE requires a mask size that exceeds hardware limits (100).	PICTURE
208	Size specification error.	PICTURE
209	Class error. Classes are NUMERIC, ALPHA, ALPHANUMERIC, or EDITED-NUMERIC.	PICTURE
	210 Group - ILLEGAL USE OF RESERVED WORD	
210	Illegal NEXT.	
210	Illegal use of reserved word.	
210	Reserved word has been used as an operand.	MOVE

Table D-2 (cont)

Compiler Error Messages

Error Message Number	Description	Verb
210	Operand has more than one subscript set.	
210	Reserved TO not allowed with GIVING.	ADD
210	Illegal use of reserved word.	
210	BY not allowed without GIVING.	DIVIDE
211	Illegal declaration at group level (e.g., PC, BZ, JUST, or SY).	
212	Illegal declaration.	
213	Illegal use of a fixed name.	
	220 Group - ILLEGAL WORD	
220	Illegal operator.	
220	Required BY missing.	MULTIPLY
220	Word ends with a hyphen.	
220	Illegal literal.	
220	Illegal word/operand/operator.	
220	Illegal word.	
220	FIRST does not follow UNTIL.	EXAMINE
220	ALL, LEADING, or FIRST not found.	EXAMINE
220	BY missing.	EXAMINE
220	TALLYING or REPLACING not found.	EXAMINE

Table D-2 (cont)

Compiler Error Messages

Error Message Number	Description	Verb
221	No ALPHA characters in a DATA DIVISION name.	
222	Word or name exceeds 30 characters.	
223	Illegal file-name after a level-number.	
223	Illegal data-name after a level-number.	
224	Reserved word required at this point.	
225	Both a label list and ALL appear in the MONITER statement. These are mutually exclusive.	
226	SELECT declaration error.	FILE-CONTROL
227	I-O CONTROL declaration is in error.	I-O-CONTROL
228	FD declaration in error.	FILE SECTION
229	Missing SELECT.	FILE-CONTROL
	<u>230 Group - INCOMPLETE STATEMENT</u>	
230	Missing TO or FROM for CORRESPONDING.	
230	MOVE Statement does not contain the word TO and/or no operand preceeding the word TO and/or no operand following the word TO.	MOVE
230	Unmatched right parenthesis.	COMPUTE
230	Statement is not complete.	EXAMINE

Table D-1 (cont)

Compiler Error Messages

Error Message Number	Description	Verb
230	Incomplete statement.	CONDITIONAL
231	Missing FROM.	SUBTRACT
239	No label after PROCEED TO.	DATA-COMM
	<u>240 Group - LEVEL ERROR</u>	
240	Missing FD after FILE-SECTION.	
241	Missing level-number.	
242	Level-number not within range of one to 99.	
243	The level of the operand in a RENAMES clause is 01, 66, 77, or 88.	RENAME
244	Erroneous level-number sequence (e.g., 02, 04, 03).	
245	REDEFINES operand has incorrect level.	
246	01 level missing.	
247	Missing level-number or indicator.	
248	Data-name must be at 77 level.	FILE-CONTROL
249	File not declared.	
	<u>250 Group - LIMIT EXCEEDED</u>	
250	Not more than two AFTERs allowed.	PERFORM

Table D-2 (cont)

Compiler Error Messages

Error Message Number	Description	Verb
251	Composite field size greater than 98 digits, or greater than 99 digits if MULTIPLY or DIVIDE.	
252	FD information table full.	
252	Operand stack full.	
252	Output limit exceeded.	
253	PC information table full.	
253	Operator stack full.	
254	Subscript stack full.	
255	Literal stack full.	
	260 Group - MISSING DECLARATIVES	
260	Missing DECLARATIVES.	
261	DECLARATIVES Missing.	
	270 Group - MISSING DIVISION/SECTION HEADER	
270	Missing DIVISION or SECTION header.	
271	The word SECTION is missing.	
272	The word DIVISION is missing.	
	280 Group - MISSING END DECLARATIVES	
280	Missing END DECLARATIVES.	

Table D-2 (cont)

Compiler Error Messages

Error Message Number	Description	Verb
	290 Group - MISSING LABEL	
290	The required procedure label immediately following the PROCEDURE DIVISION is missing.	
291	No label after a SECTION name, GO TO, or EXIT paragraph.	
292	Missing implied label after GO TO, PERFORM, etc.	LABEL
	300 Group - MISSING OPERAND	
300	Less than two operands.	
300	No operand after GIVING.	
301	Missing ASSIGN clause in a SELECT statement.	FILE-CONTROL
303	Missing FILE ID value.	FILE SECTION
304	Hardware name missing.	FILE-CONTROL
305	Illegal MULTI-FILE ID.	FILE-CONTROL
307	THRU missing in the FILE-LIMITS clause.	FILE-CONTROL
	310 Group - MISSING PERIOD	
310	A required period is missing.	
	320 Group - ENTER SYMBOLIC	
320	ENTER SYMBOLIC element missing or wrong.	

Table D-2 (cont)

Compiler Error Messages

Error Message Number	Description	Verb
	330 Group - NO CORRESPONDING ITEMS	
330	CORRESPONDING left rejected because CORRESPONDING right failed data-name qualifications.	
331	Missing CORRESPONDING right.	
332	CORRESPONDING right rejected because CORRESPONDING left entry failed on data-name qualification.	
333	CORRESPONDING left and right same data-name.	
334	CORRESPONDING ranges overlap.	
335	One or both CORRESPONDING data-names is a FD.	
336	Both CORRESPONDING data-names are not group items.	
338	CORRESPONDING pairs or CORRESPONDING items do not have the same qualification.	
338	No CORRESPONDING pairs.	
339	No CORRESPONDING logic attempted. Data structure is bad (usually duplicate data-names) in one or both CORRESPONDING ranges.	

Table D-2 (cont)

Compiler Error Messages

Error Message Number	Description	Verb
	340 Group - PICTURE ERROR	
340	PICTURE string contains more than 30 characters.	
341	PICTURE missing in elementary item.	
342	BLANK WHEN ZERO class is ALPHABETIC or ALPHANUMERIC. Change PICTURE to NUMERIC or NUMERIC EDITED.	
343	Justify class is NUMERIC or NUMERIC-EDITED. Change to ALPHA or ALPHANUMERIC.	
347	The reserved word MOD may be used only at the 01 level and only in working-storage.	
	350 Group - QUALIFICATION ERROR	
350	No data-name qualifier after OF or IN.	
351	Unidentified implicit label or qualifier.	
351	Unidentified implicit data-name or qualifier.	
352	Duplicate paragraph-names in same SECTION.	
353	Illegal qualification.	
354	SECTION name used as qualifier not unique.	

Table D-2 (cont)

Compiler Error Messages

Error Message Number	Description	Verb
355	Insufficient qualification to determine uniqueness of label.	
355	Insufficient qualification to determine uniqueness of data-name.	
356	Label qualifier not a SECTION name.	
357	Too many labels in the program; request more core.	
357	More than 10 levels of qualification, or too many data-names in the program. If too many data-names, request more core.	
	360 Group - REDEFINES SIZE ERROR	
360	RENAMES size error.	
361	REDEFINES name error.	
362	REDEFINES size error.	
363	RENAMES group has odd address or odd size.	
	370 Group - REFERENCE FORMAT ERROR	
370	Reference format error.	
	380 Group - SUBSCRIPT ERROR	
380	RENAME operand is subscripted.	RENAME
380	More than three subscripts.	

Table D-2 (cont)

Compiler Error Messages

Error Message Number	Description	Verb
380	Missing subscript.	
380	Subscript has scale.	
380	Too many subscripts.	
381	OCCURS has been specified on the 01 or 77 level entry.	
381	More than three subscripts.	
382	OCCURS operand specification error.	
383	REDEFINES operand subscripted.	
384	Missing right parenthesis.	
	<u>390 Group - UNIDENTIFIED WORD</u>	
390	Unidentified word.	
390	Unidentified input.	
390	Unidentified input.	
	<u>400 Group - USAGE ERROR</u>	
400	COMPUTATIONAL USAGE has been specified for a display device (e.g., card reader, printer, or punch).	
401	COMPUTATIONAL specified on an item with a non-numeric PICTURE.	

Table D-2 (cont)

Compiler Error Messages

Error Message Number	Description	Verb
402	Group USAGE conflicts with USAGEs of elementary items within that group.	
	<u>410 Group - VALUE ERROR</u>	
410	VALUE specified for REDEFINED area.	
411	VALUE operand is missing.	
412	VALUE has been specified for a subscripted item.	
413	VALUE has already been specified at the group level.	
419	VALUE may not be a signed literal if PICTURE specifies J or K sign.	
	<u>420 Group - ILLEGAL OPERATOR</u>	
429	Illegal word.	DATA-COMM
	<u>430 Group - COMPILE ERROR</u>	
431	Could not find data item in the data-name information. Usually due to some other syntax error or may indicate software/ hardware failure.	
431	Illegal type code. Indicates software/ hardware failure.	
431	No such scan class.	
432	No such scan class.	

Table D-2 (cont)

Compiler Error Messages

Error Message Number	Description	Verb
434	No such type.	
	440 Group - MAY NOT REFERENCE OUTSIDE DECLARATIVES	
440	Data-names declared as being within a LABEL RECORD may be referenced only within the declaratives.	
	450 Group - CONDITION NAME CONSTRUCT ERROR	
450	Bad condition-name construct.	CONDITION-NAME
	460 Group - FILLER CANNOT BE A GROUP ITEM	
460	A group item may not be named FILLER.	
	470 Group - TOO MANY DATA NAME MONITER ENTRIES	
470	Trying to MONITOR too many data-names.	
	480 Group - EXCEEDED DATA NAME LIMIT IN COMPILATION	
480	Program contains too many data-names for the amount of core assigned to the COBOL compiler. Recompile, using a core card to make more core available to the compilation.	
	490 Group - ILLEGAL CONDITION-PRIMARY	
490	The primary is not boolean.	
	500 Group - ILLEGAL OPERATOR IN CONDITION-PRIMARY	
500	Bad operator.	

Table D-2 (cont)

Compiler Error Messages

Error Message Number	Description	Verb
	510 Group - ILLEGAL QUANTITY AT START OF CONDITION	
510	Illegal quantity.	
	520 Group - MISSING RIGHT PARENTHESIS	
520	Matching right parenthesis.	
	530 Group - THE RELATION MUST BE COMPLETE	
530	The relation is incomplete.	
	540 Group - AN ARITHMETIC EXPRESSION CANNOT BE NEGATED	
540	Expression cannot be negated.	
	550 Group - MISSING LOGICAL CONNECTIVE	
550	Logical connective missing.	
	560 Group - ILLEGAL CONDITION	
560	Illegal condition.	
561	Index-data-name versus a literal or normal data-name.	
	570 Group - MISPLACED WORD-NEXT	
570	Reserved word NEXT misplaced.	IF STATEMENT
	580 Group - SUBJECT AND OBJECT ARE LITERALS	
580	Both operands may not be literals.	CONDITIONAL
	590 Group - USAGE MUST BE DISPLAY, NON-NUMERIC COMPARES	
590	The USAGE must be DISPLAY unless both of the operands are NUMERIC.	CONDITIONAL

Table D-2 (cont)

Compiler Error Messages

Error Message Number	Description	Verb
	600 Group - NUMERIC VERSUS ALPHA COMPARE IS ILLEGAL	
600	Cannot compare a NUMERIC item to an ALPHA item.	CONDITIONAL
	610 Group - COMPOSITE OPERAND SIZE MUST BE LESS THAN 100	
610	Compare of COMPUTATIONAL items restricted to less than 100 digits.	CONDITIONAL
	620 Group - CLASS TEST OPERAND MUST BE DISPLAY	
620	USAGE must be DISPLAY.	CONDITIONAL
	630 Group - FILLER IS FOLLOWED BY 88 LEVEL ENTRY	
630	May not have a 88 level on a FILLER.	
	640 Group - USE: AFTER BEGINNING OR BEFORE ENDING	
640	Statement should read: USE AFTER . . . BEGINNING . . . or USE BEFORE . . . ENDING. . .	USE
	650 Group - MAXIMUM: FILE CONTAINS 20 BY 99999999 RECORDS	
650	DISK files may not contain more than the equivalent of 5,000,000 20-character records in a single file page. There may not be more than 2G pages per file.	FILE SECTION
	660 - Group - INVALID USE OF PERFORM STATEMENT	
660	Object program segment structure incorrect and would result in overlaying the PERFORM.	PERFORM
	670 Group - FILE CONTAINS CLAUSE REQUIRED FOR DISK	
670	Required FILE CONTAINS clause has been omitted.	FILE SECTION

Table D-2 (cont)

Compiler Error Messages

Error Message Number	Description	Verb
	680 Group - ACTUAL KEY/FILE LIMIT IS NOT PC 9(8) CMP	
680	Data-names defined as ACTUAL-KEYs or FILE-LIMITS must be declared PC 9(8) CMP.	
	690 Group - PC TABLE FULL: REQUEST MORE CORE	
690	Table limit has been exceeded. Recompile the program using a core card to make more memory available, or reduce the number of unique PICTUREs.	
	700 Group - ACTUAL KEY REQUIRED FOR RANDOM FILES	
700	An ACTUAL-KEY must be defined for all RANDOM files.	FILE-CONTROL
	710 Group - LABEL MUST BE FIRST 01 OF FD	
710	If a LABEL defined for referencing within the DECLARATIVES, it must be the first 01 following the FD.	FILE SECTION
	720 Group - BACKUP IS FOR PRINTER OR PUNCH ONLY	
720	When backup is specified in a SELECT clause, that file must be assigned to the printer or punch.	FILE-CONTROL
	730 Group - MISSING USE SENTENCE	
730	The USE sentence must immediately follow each SECTION in the DECLARATIVES.	
	740 Group - MISSING SECTION NAME	
740	A SECTION name must follow DECLARATIVES.	

Table D-2 (cont)

Compiler Error Messages

Error Message Number	Description	Verb
	760 Group - FILE MUST BE A SORTER	
760	Specified declarations for sorter file only.	I-O-CONTROL
	770 Group - SORTER RECORD MUST BE 100 CHARACTERS	
770	FD must describe 100 characters.	FILE SECTION
	780 Group - GROUP ADDRESS NOT MOD 2	
780	If an elementary CMP item not character adjusted is redefined as a group (or alphanumeric) item that must be character adjusted.	
	790 Group - LISTER ACTUAL KEY	
790	ACTUAL KEY for LISTER is not PC 9(4) CMP.	
	800 Group - INDEX DECLARATION	
800	INDEX data item declaration error.	
	810 Group - VALUE CLAUSE	
810	VALUE specified for REDEFINES area.	
	820 Group - PAPER TAPE	
820	Paper tape file declared as being BLOCKED.	
	830 Group - RENAMING	
830	RENAMES operand range error.	
	840 Group - SD DEFINITIONAL ERROR	
840	SD file must be SEQUENTIAL and have no declared FILE-LIMITS.	

Table D-2 (cont)

Compiler Error Messages

Error Message Number	Description	Verb
	850 Group - VARIABLE LENGTH DISK RECORDS	
850	Variable length disk records are not allowed.	
	860 Group - DATA COMM. ACTUAL KEY	
860	Data Communication ACTUAL KEY IS NOT defined PC 9(6) CMP.	

APPENDIX E

ENTER SYMBOLIC RESERVED WORDS

ADD

AND

BAS

BCT

BOT

BUN

BZT

CPA

CPN

DEC

DIV

DSP

EDT

EQL

EXT

FAD

FDV

FMP

FSU

GEQ

GTR

HBK

HBR

INA

INC

IX1

IX2

IX3

LEQ

LIT

LSS

MPY

MVA

MVC

MVL

MVN

MVR

MVW

NEQ

NOP

NOT

NTR

OFL

ORR

SDE

SDU

SEA

SGN

SMF

SUB

SZE

SZU

TRN

UNS

* See special instructions, page 1-10.

APPENDIX F

DISK FILE HANDLING

GENERAL.

Disk space for files within the MCP controlled environment is allocated to a program, dynamically, and addressed by the program by record location within file-name. Record location is defined as one (1) through N, where N equals the number of records contained in the file. This may be further redefined by the FILE-LIMITS clause. The programmer can further define his file, for disk allocation purposes, as occupying multiple areas of disk to be allocated as needed, i.e., when the previous area is full or the program attempts to access a record, to WRITE, within an unallocated area. These areas of disk records can be scattered across different Electronic Units at the option of the programmer or anywhere on the disk, where adequate space for that area is available, at the option of the MCP. This area allocation scheme in no way affects the record addressing concept in the object program nor adds to the READ/WRITE time within the MCP.

It becomes obvious, therefore, that for this allocation scheme to be possible the MCP must know the status of the Disk File at all times. There are two tables maintained by the MCP on disk that are used to represent this status. They are the Disk Directory and the Available Disk List.

The Disk Directory contains information concerning all files and programs currently in the disk library. This data (name, physical disk address of each area currently allocated, number of records per area, End-of-File pointer, etc.) in concert with the File Information Block (FIB) located within the object program using that file, contains all the data necessary for the MCP to access records at the request of the object program.

The Available Disk List is an MCP maintained linked list containing, by ascending disk address, all available disk space and their sizes in physical segments (100 bytes/segment).

When an area of disk is requested by a program, the MCP searches the Available Disk List for an area of disk large enough to contain the desired records. If an area of disk large enough is not found, the program making the request is suspended and the operator is notified of NO USER Disk. Assuming an area of sufficient size is found, this area is removed from the Available Disk List and is a candidate for entry in the Disk Directory depending on other actions taken by the program (these other actions such as OPEN and CLOSE will be discussed later). Depending on these actions,. that area of disk is either returned to the Available Disk List or entered in the Disk Directory as a permanent file. If that area is returned to the Available Disk List, it is subject to reuse as often as needed.

If the file is made permanent, the Disk Directory is searched for a matching file-name. If a match is not found, the new file is entered in the Disk Directory and is available to be opened by another program. Should a matching file-name be found, the operator is notified of this duplicate and the program is suspended until a proper operator response is made. There are three options available to the operator: remove ((MX)RM) the old file, change the name of the old file, and discontinue ((MX)DS) the program thus returning the disk area for that file to the Available Disk List.

If a file is OPEN OUTPUT in a sequential mode, records are written to the file starting at record one (1) and proceeding until the program CLOSEs that file or the definition of that file (as defined by the FILE CONTAINS and the FILE-LIMITS clauses) is reached. When the first area of disk is filled, in a multi-area defined file, the next area is obtained. This is continued until the End-of-File condition is reached. It can be seen from this description that in the sequential mode, areas are allocated and used beginning with area one (1) and proceeding in sequence until the last area is allocated. The maximum number of areas allowed in the FILE CONTAINS clause is twenty (20).

If a multi-area file is OPEN OUTPUT or O-I in a random mode, records are written to this file in accordance to a record number (ACTUAL

KEY) supplied by the object program dynamically. Since the actual key for writing records is dynamic and not necessarily sequential, it is possible that the areas (pages) of disk allocated may be neither sequential nor consecutive. For example, areas three (3), five (05), and seven (07) may be allocated to a file if the actual key at record WRITE time placed records in only these areas. In this example, areas 01, 02, 04, and 06 have not been allocated to this file, but the End-of-File pointer located in the Disk Directory for this file would contain the value of the highest record number written (located in area 07).

The above described disk allocation scheme is paramount to understanding the other areas of disk file usage. These areas are:

a. OPEN sequential.
b. Open Random.
c. CLOSE.
d. READ, WRITE, and SEEK with RANDOM DISK files.

OPEN SEQUENTIAL.

This OPEN consists of three types: OPEN INPUT, OPEN OUTPUT, and OPEN I-O.

OPEN OUTPUT. This type of OPEN is for new files (the file does not exist or is to be replaced in the Disk Directory by this entry). Records are written sequentially beginning with record one (1) and ending when the program CLOSEs that file or the maximum record as defined in the FILE CONTAINS clause is reached. Disk will not be obtained until the first write is attempted and then only the first area is obtained.

If an attempt is made to write past the last record defined in the FILE CONTAINS clause, the at end branch will be taken on the WRITE statement. Absence of an AT END will cause a program termination (EOF NO LABEL).

OPEN INPUT. This OPEN requires the existence of the file in the Disk Directory (an old file). Records are made available to the program starting with record number 1, or as defined by the FILE-LIMITS clause. The End-of-File branch is taken on a READ statement when an attempt is made to access past the End-of-File pointer, or the end record of the FILE-LIMITS clause. The End-of-File branch will also be taken if an attempt is made to READ a record from an unallocated area, even if the record number is below the End-of-File pointer. Again, absence of an AT END will cause an EOF NO LABEL termination.

OPEN I-O. This OPEN requires the file to be on disk with an entry in the Disk Directory. This method allows users to update a sequential file. The following examples show mixing READ and WRITE commands:

a. READ record-1 READ record-2 ... .
b. READ record-1 WRITE record-1 READ record-2 ... .
c. WRITE record-1 WRITE record-2 ... .
d. WRITE record-1 READ record-2 WRITE record-2 ... .

OPEN RANDOM.

This OPEN consists of four types: OPEN OUTPUT, OPEN INPUT, OPEN INPUT-OUTPUT, and OPEN OUTPUT-INPUT. There are also two options that can be used with this type OPEN: OPEN WITH LOCK option and OPEN WITH LOCK ACCESS option.

OPEN OUTPUT. This type of OPEN is for new files. Records are written to this file in accordance to the value of the ACTUAL KEY, the contents of which is dynamic throughout the execution of the program. A new area of disk is allocated the first time an attempt is made to WRITE a record into that area. Any attempt to exceed the bounds of the file, as declared in the FILE CONTAINS clause, will invoke the invalid key branch off the WRITE statement. The random writing of either blocked or unblocked records is provided. The only I/O statements allowed are WRITE and SEEK.

OPEN INPUT. This type of OPEN assumes the existence of an old file. Records are read in accordance to the value of the ACTUAL KEY, the contents of which are dynamically provided by the program at object time. Any attempt to exceed the End-of-File pointer for that file, access a record in an unallocated area, or violate the FILE-LIMITS clause will invoke the INVALID KEY branch of the READ statement. Random reading of blocked or unblocked records is provided. The two I/O statements allowed are READ and SEEK.

OPEN INPUT-OUTPUT (I-O). This OPEN technique for random files incorporates the capabilities of both OPEN INPUT and OPEN OUTPUT. This OPEN type assumes an old file (one that exists on the Disk Directory). Files OPENed in this manner are generally OPEN for update, that is READ record-n update record-n in core, WRITE record-n in its updated form. END-OF-FILE options for this type of OPEN are identical to the two previous types. INVALID KEY on the READ is when the End-of-File pointer is exceeded or a READ attempt within an unallocated area is exceeded. INVALID KEY on the WRITE is when maximum file declaration is exceeded. Any attempt to access a record within an unallocated area for writing will cause that area to be allocated and all records within that area, not exceeding End-of-File pointer, are available to the program, regardless of their validity.

OPEN OUTPUT-INPUT (O-I). This type of OPEN for random files is identical to OPEN I-O with the exception that the file is assumed to be a new file (one that does not exist in the Disk Directory). This OPEN option will shortcut the usual method of using I-O work files within a program. This procedure is OPEN OUTPUT, WRITE records, CLOSE, OPEN I-O, etc. This option does not, nor was it intended to, replace OPEN I-O since the use of OPEN O-I assumes a new file each time.

Since INVALID KEY or AT END is based on the End-of-File,pointer, it is possible to read records that were not created by this program. For example; a disk file is declared by a program as containing one by 100 records (FILE CONTAINS clause). The file is OPEN OUTPUT in a random mode and the program sets the ACTUAL KEY for that file at 100 and WRITES a record. If the program CLOSES that file in a manner which causes it to be entered in the Disk Directory, area 1 will be allocated with the End-of-File pointer in the Disk File Header set at 100. If another program OPENS that file INPUT and attempts to access record number 1, the first record will be made available. In this example, it is possible for the program to READ 99 garbage records before the first valid record. An attempt to access beyond 100 will cause an INVALID KEY or AT END branch to be taken off the READ statement.

OPEN WITH LOCK. The use of the LOCK option with any OPEN type will prohibit any other program from OPENing that file in any manner until the program causing the LOCK releases the file.

OPEN WITH LOCK ACCESS. The use of the LOCK ACCESS option with any OPEN will prohibit any other program from OPENing that file except for INPUT. This option allows only one update program (I-O) but any number of read programs (INPUT) to access the file. The LOCK ACCESS remains in effect until the program causing this releases the file.

CLOSE.

CLOSE consists of four types: CLOSE file-name, Terminate CLOSE, CLOSE PURGE, and CLOSE LOCK or CLOSE RELEASE.

CLOSE FILE-NAME. This is normally considered a temporary CLOSE option and may cause the programmer some difficulty if not understood. If the program goes to EOJ after this CLOSE type on a new file (OPEN OUTPUT or O-I), the file is not entered in the Disk Directory and the area(s) occupied by this file is returned to the Available Disk List. If the file is an old file (one previously entered in the Disk Directory), the Disk File Header for that file is updated with its current status and written back to the Disk Direcotry thus causing the new records to be permanently assigned.

TERMINATE CLOSE. This type of CLOSE is an implicit MCP CLOSE of the file caused by a premature termination of the program due to program failure or absence of an explicit CLOSE by the program prior to EOJ. The MCP actions for this type are identical to the plain CLOSE described above.

CLOSE PURGE. This type of CLOSE will cause the MCP to remove the file (old or new) and return the disk areas described for that file to the Available Disk File. The file no longer exists in the Disk Directory.

CLOSE LOCK or CLOSE RELEASE. These CLOSE types cause new files (OPEN OUTPUT or O-I) to be made permanent by entering the file in the Disk Directory. This is done by the action of writing the Disk File Header, created and updated by the MCP during execution of this program, into the Disk Directory and entering the file-name in the Disk Directory List. The action taken for an old file (one that exists in the Disk Directory) is identical to the plain CLOSE type.

It is possible to override the action of CLOSE or Terminate CLOSE as described for new files (files not currently in the Disk Directory) by use of the SAVE option in the SELECT clause. For example,

```
SELECT file-name ASSIGN TO DISK SAVE . . . .
```

causes new files to be entered in the Disk Directory when a program failure causes an unexpected termination of the program. The file and End-of-File pointer will reflect the status of the file at the moment of termination.

READ, WRITE, AND SEEK WITH RANDOM DISK FILES.

Before any further discussion on this subject, definitions of READ, WRITE, and SEEK as related to random disk files will be helpful.

a. READ - make available to the object program a logical record addressed by the setting of the ACTUAL KEY.

b. WRITE - release a logical record addressed by the ACTUAL KEY to the MCP for output to the disk.

c. SEEK - fill a program buffer with a block of records containing the record pointed to by the ACTUAL KEY if the file is blocked, or cause the record required, if unblocked, to be read. If the MCP has the required record in a program buffer, by virtue of a previous SEEK, a new physical READ will not be initiated.

The reading and writing of logical records is an object program concept. The initiation of physical READs and WRITEs is an MCP concept. Whether a file is blocked or unblocked, in no way affects the object program concept of logical READ and WRITE, the burden of blocking and unblocking is on the MCP. Since a physical READ or WRITE to a disk file must begin at the start of a segment address and be in multiples of 100 byte segments, and, no record size or block size restrictions are made on the object program, the possibility that blocked records may overlap segment addresses exists. If the MCP initiated a WRITE operation under these conditions, adjacent records within the block could be destroyed. To keep this from occurring, the MCP must READ the block containing the record to be written, move the new record into the block, and then WRITE the entire block back in the same location.

The use of a SEEK prior to a WRITE would accomplish this action. However, if an explicit SEEK is not given prior to a WRITE of blocked records, an implicit SEEK will be done by the MCP to load the block prior to the WRITE. A SEEK is never needed on files declared OUTPUT, since the MCP will handle any SEEK requirements when the writing of records would cause adjacent record destruction.

On files OPENed INPUT, the absence of an explicit SEEK prior to a READ will cause the MCP to initiate an implicit SEEK to load the required record. Therefore, a SEEK is never required explicitly in files OPENed INPUT. The use of explicit SEEKs may create a speed increase in anticipation of random READs.

The only condition which requires the object program to explicitly SEEK file-records is when USE FOR KEY CONVERSION is used. This declarative will only be executed during an explicit SEEK. Implicit SEEKs will not cause this action.

The following is a summary of the use of SEEK:

a. Only valid in random files.

b. Never required explicitly in random files.

c. Required if USE FOR KEY CONVERSION is desired.

d. Will improve speed in anticipation of random reads with multiple buffers.

NOTE

> A special note for those who are familiar with our previous implementation of random disk files on both the B 5500 and B 3500 regarding the setting of the ACTUAL KEY. Henceforth, a READ or WRITE of a record will always use the value of the programs ACTUAL KEY at the time of actual I/O initiation. No assumptions will be made by the MCP on WRITE after READ as in the past.

SPECIAL DISK CONSIDERATIONS.

Many problems have been encountered by the B 2500/B 3500 programmers regarding the use of disk files. These problems, caused by violations of disk file usage, have created a number of Disk Directory failures, run-time object program failures, operational difficulties, and many hours of trouble shooting. The following is a list of violations and their effects. Also, there are changes to COBOL syntax which may cause a program which previously was error free to fail syntax or the latest COBOL Compiler.

a. It is now a syntax error to omit an INVALID KEY statement on a READ of a random disk file.

b. When the FILE-LIMITS clause is involved, it is a syntax error if a WRITE statement does not contain an INVALID KEY or AT END statement.

c. Run-time failures can occur if an alternate area (buffer and work area) is not defined for random disk files. Since the MCP keeps track (by actual disk addresses) of the records it has read by virtue of previous SEEK operations, destruction of a record by the object program in a buffer (buffer only technique) and subsequent request for a READ of that record will cause the MCP to "Logically Read" the same record, i.e., return control to the program pointing to the same destroyed record. This will only be true if an intervening I/O has not changed the actual disk addresses maintained by the MCP for that file in the FIB (File Information Block).

d. It is invalid to have more than one file utilizing the SAME AREA OPEN at the same time. A run time -- INV FILE OPEN message, and program termination will occur if a program attempts to OPEN a file that shares buffers without first causing a CLOSE RELEASE or CLOSE LOCK of the sharing file.

e. It is invalid to change the name of a file by the program after it is OPEN. An attempt to CLOSE a file by a name different than when it was OPENed will result in program termination and an -- INV FILE CLOSE.

f. An -- INV CONTROL INST message will be displayed if a program attempts to ZIP a name change to a file that is currently in use by any program.

g. An attempt to purge or remove a file in use by any program will cause an appropriate response and be disallowed. A remove request will yield an -- INV CONTROL INST and a CLOSE PURGE by a program when that file is in use will yield a ** LOCKED FILE message and suspension of the offending program until the file is no longer in use.

NOTE

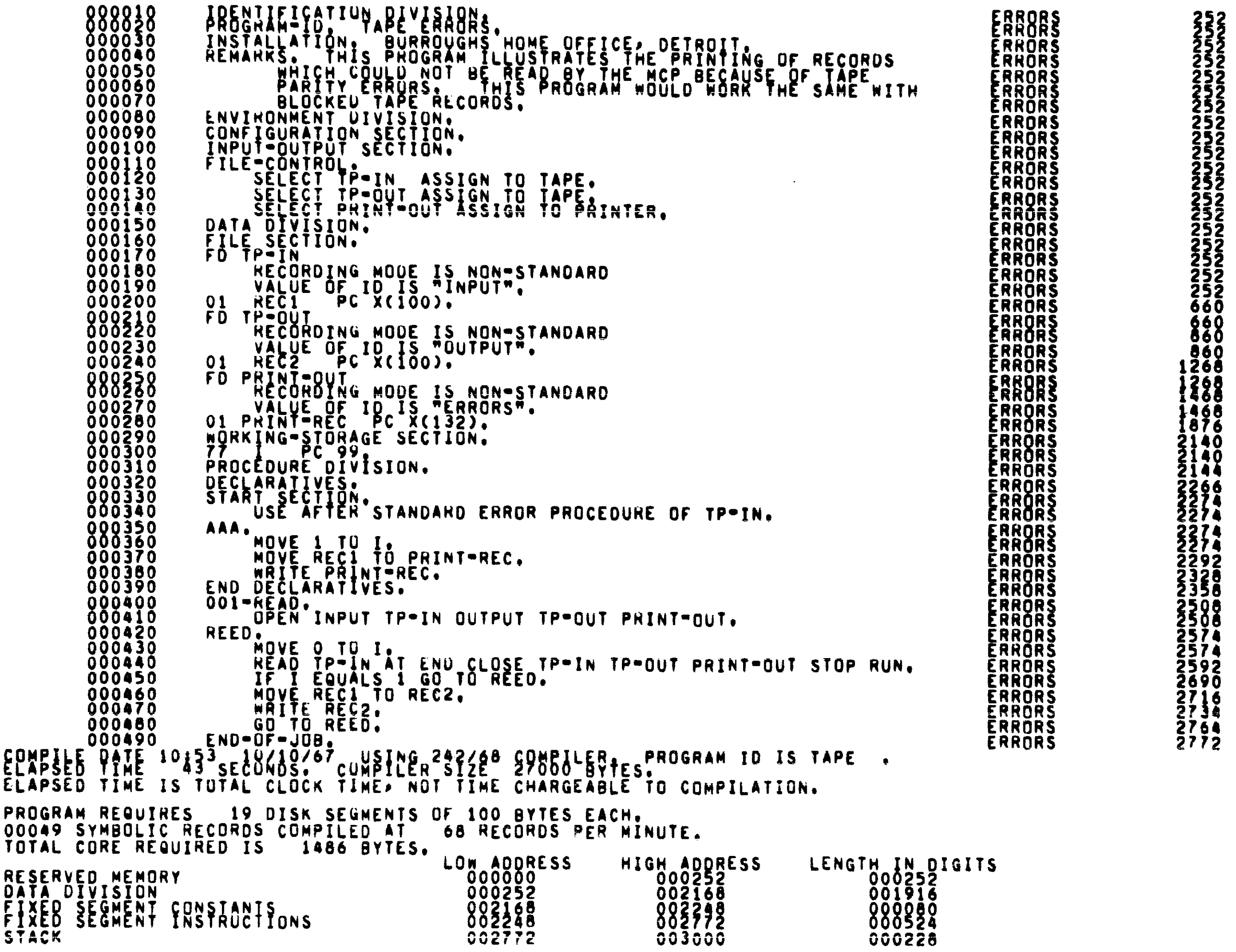
A straight CLOSE is not sufficient to eliminate these results since

NOTE (cont)

buffer release, I/O assignment table linkage, and lock file maintenance is only affected by use of RELEASE or LOCK in the CLOSE statement.

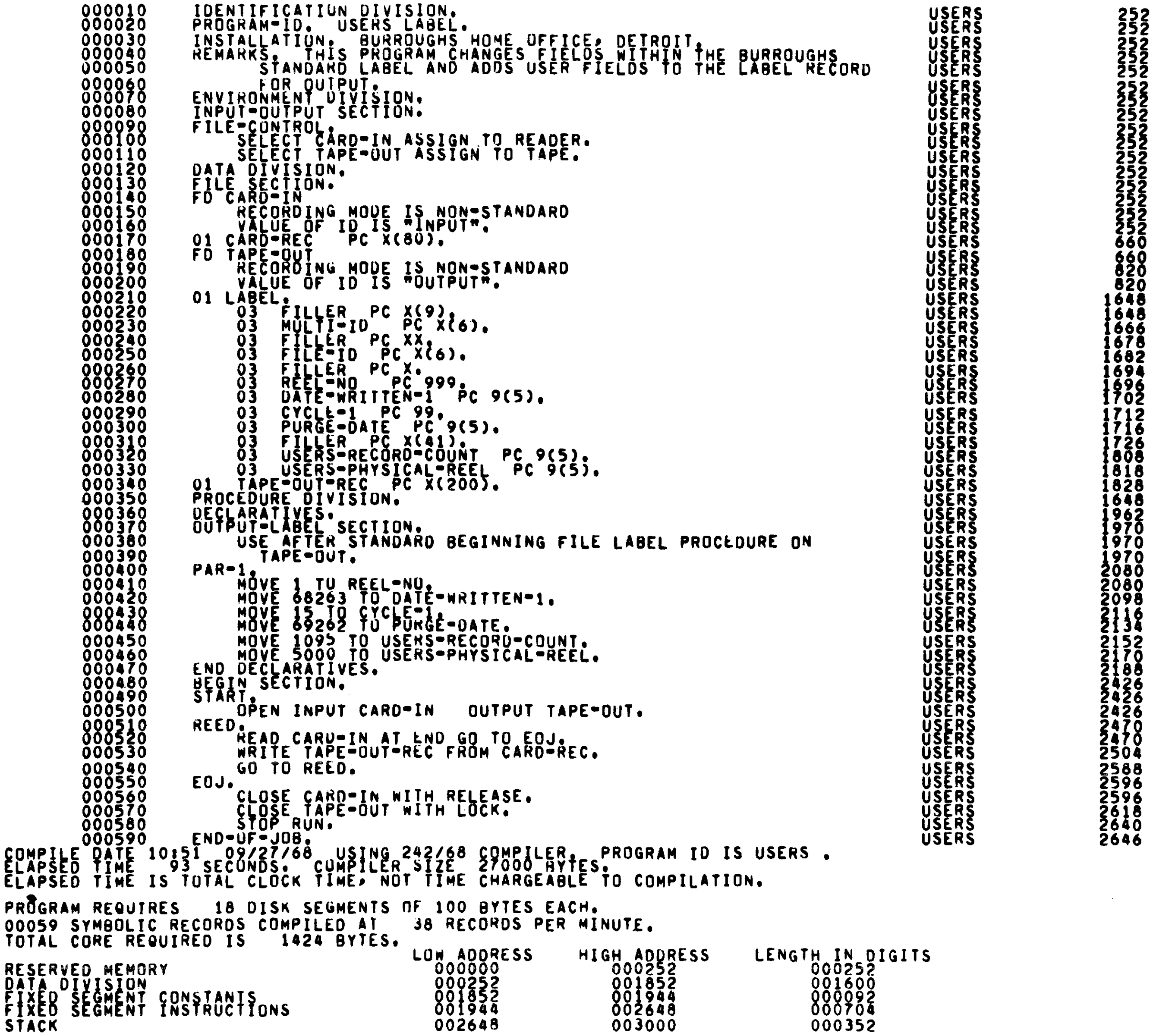
APPENDIX G
PROGRAMING EXAMPLES

The examples contained in this appendix are for reference or educational purposes and are not to be construed as being the only method possible to obtain a given result. Experience in the COBOL language is greatly enhanced by referencing simple programs or portions of a program and progresses at the rate of a programmer's perception of given problems. This appendix attempts to relate certain elements of the language so that a programmer can quickly perceive the entire spectrum of the rules and usage of the COBOL syntax.



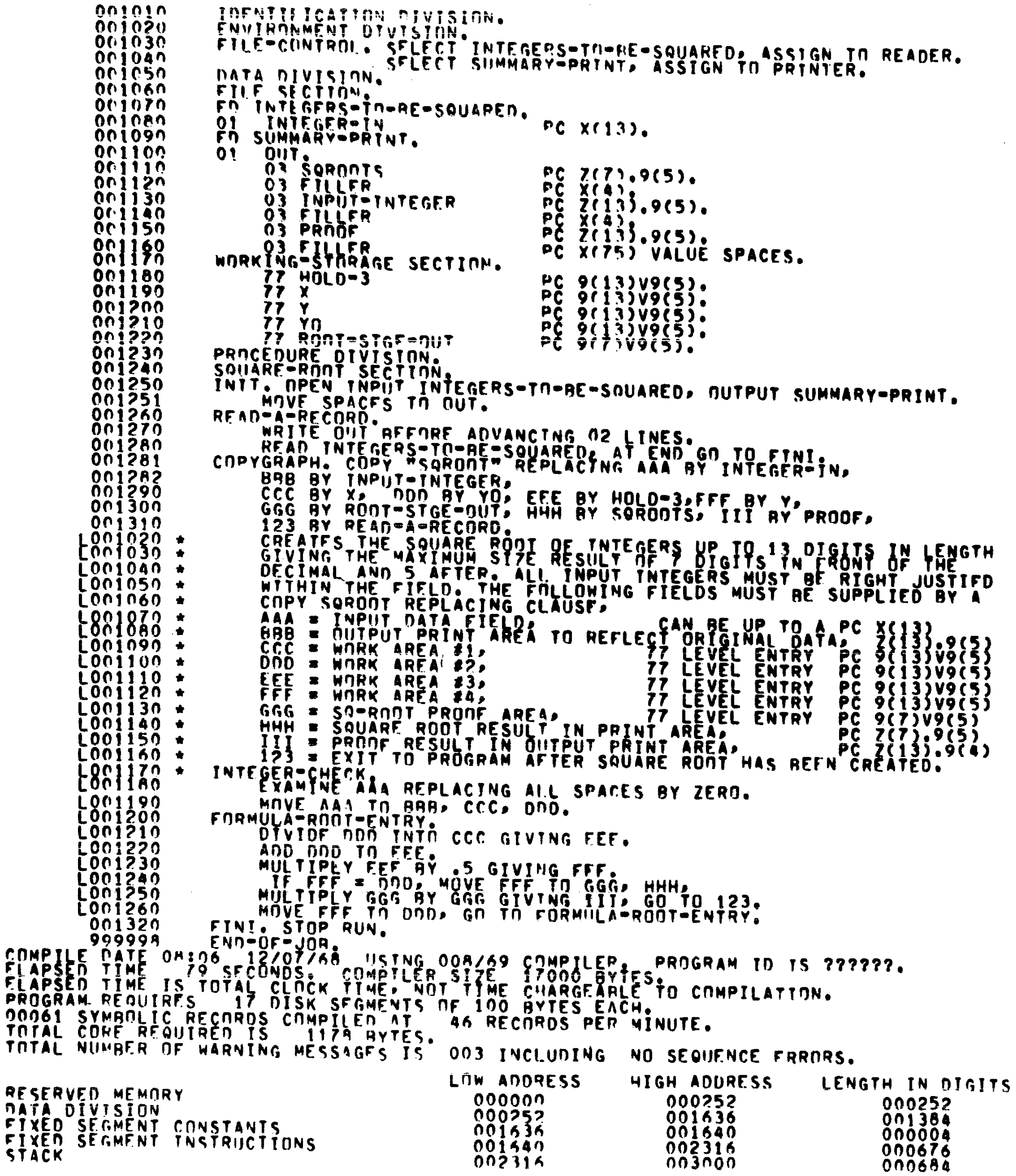
```
      000010     IDENTIFICATION DIVISION.                                      ERRORS         252
      000020     PROGRAM-ID.  TAPE ERRORS.                                     ERRORS         252
      000030     INSTALLATION.  BURROUGHS HOME OFFICE, DETROIT.                ERRORS         252
      000040     REMARKS.  THIS PROGRAM ILLUSTRATES THE PRINTING OF RECORDS    ERRORS         252
      000050             WHICH COULD NOT BE READ BY THE MCP BECAUSE OF TAPE    ERRORS         252
      000060             PARITY ERRORS.   THIS PROGRAM WOULD WORK THE SAME WITH ERRORS        252
      000070             BLOCKED TAPE RECORDS.                                 ERRORS         252
      000080     ENVIRONMENT DIVISION.                                         ERRORS         252
      000090     CONFIGURATION SECTION.                                        ERRORS         252
      000100     INPUT-OUTPUT SECTION.                                         ERRORS         252
      000110     FILE-CONTROL.                                                 ERRORS         252
      000120          SELECT TP-IN  ASSIGN TO TAPE.                            ERRORS         252
      000130          SELECT TP-OUT ASSIGN TO TAPE.                            ERRORS         252
      000140          SELECT PRINT-OUT ASSIGN TO PRINTER.                      ERRORS         252
      000150     DATA DIVISION.                                                ERRORS         252
      000160     FILE SECTION.                                                 ERRORS         252
      000170     FD TP-IN                                                      ERRORS         252
      000180          RECORDING MODE IS NON-STANDARD                           ERRORS         252
      000190          VALUE OF ID IS "INPUT".                                  ERRORS         252
      000200     01  REC1    PC X(100).                                        ERRORS         660
      000210     FD TP-OUT                                                     ERRORS         660
      000220          RECORDING MODE IS NON-STANDARD                           ERRORS         860
      000230          VALUE OF ID IS "OUTPUT".                                 ERRORS         860
      000240     01  REC2    PC X(100).                                        ERRORS        1268
      000250     FD PRINT-OUT                                                  ERRORS        1268
      000260          RECORDING MODE IS NON-STANDARD                           ERRORS        1468
      000270          VALUE OF ID IS "ERRORS".                                 ERRORS        1468
      000280     01 PRINT-REC  PC X(132).                                      ERRORS        1876
      000290     WORKING-STORAGE SECTION.                                      ERRORS        2140
      000300     77  I   PC 99.                                                ERRORS        2140
      000310     PROCEDURE DIVISION.                                           ERRORS        2144
      000320     DECLARATIVES.                                                 ERRORS        2266
      000330     START SECTION.                                                ERRORS        2274
      000340          USE AFTER STANDARD ERROR PROCEDURE OF TP-IN.             ERRORS        2274
      000350     AAA.                                                          ERRORS        2274
      000360          MOVE 1 TO I.                                             ERRORS        2274
      000370          MOVE REC1 TO PRINT-REC.                                  ERRORS        2292
      000380          WRITE PRINT-REC.                                         ERRORS        2328
      000390     END DECLARATIVES.                                             ERRORS        2358
      000400     001-READ.                                                     ERRORS        2508
      000410          OPEN INPUT TP-IN OUTPUT TP-OUT PRINT-OUT.                ERRORS        2508
      000420     REED.                                                         ERRORS        2574
      000430          MOVE 0 TO I.                                             ERRORS        2574
      000440          READ TP-IN AT END CLOSE TP-IN TP-OUT PRINT-OUT STOP RUN. ERRORS        2592
      000450          IF I EQUALS 1 GO TO REED.                                ERRORS        2690
      000460          MOVE REC1 TO REC2.                                       ERRORS        2716
      000470          WRITE REC2.                                              ERRORS        2734
      000480          GO TO REED.                                              ERRORS        2764
      000490     END-OF-JOB.                                                   ERRORS        2772
COMPILE DATE 10:53  10/10/67  USING 242/68 COMPILER.  PROGRAM ID IS TAPE   .
ELAPSED TIME    43 SECONDS.  COMPILER SIZE  27000 BYTES.
ELAPSED TIME IS TOTAL CLOCK TIME, NOT TIME CHARGEABLE TO COMPILATION.

PROGRAM REQUIRES   19 DISK SEGMENTS OF 100 BYTES EACH.
00049 SYMBOLIC RECORDS COMPILED AT   68 RECORDS PER MINUTE.
TOTAL CORE REQUIRED IS   1486 BYTES.
                                       LOW ADDRESS    HIGH ADDRESS    LENGTH IN DIGITS
RESERVED MEMORY                          000000         000252            000252
DATA DIVISION                            000252         002168            001916
FIXED SEGMENT CONSTANTS                  002168         002248            000080
FIXED SEGMENT INSTRUCTIONS               002248         002772            000524
STACK                                    002772         003000            000228
```

Figure G-1. Example of a Tape Parity Errors Routine



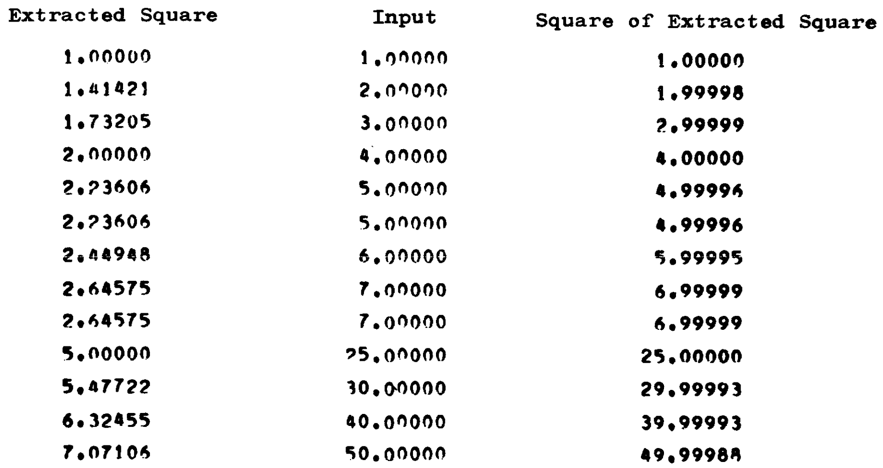
```
       000010     IDENTIFICATIUN DIVISION.                                        USERS             252
       000020     PROGRAM-ID.  USERS LABEL.                                       USERS             252
       000030     INSTALLATION.  BURROUGHS HOME OFFICE, DETROIT.                  USERS             252
       000040     REMARKS.  THIS PROGRAM CHANGES FIELDS WITHIN THE BURROUGHS      USERS             252
       000050           STANDARD LABEL AND ADDS USER FIELDS TO THE LABEL RECORD   USERS             252
       000060           FOR OUTPUT.                                               USERS             252
       000070     ENVIRONMENT DIVISION.                                           USERS             252
       000080     INPUT-OUTPUT SECTION.                                           USERS             252
       000090     FILE-CONTROL.                                                   USERS             252
       000100         SELECT CARD-IN ASSIGN TO READER.                            USERS             252
       000110         SELECT TAPE-OUT ASSIGN TO TAPE.                             USERS             252
       000120     DATA DIVISION.                                                  USERS             252
       000130     FILE SECTION.                                                   USERS             252
       000140     FD CARD-IN                                                      USERS             252
       000150         RECORDING MODE IS NON-STANDARD                              USERS             252
       000160         VALUE OF ID IS "INPUT".                                     USERS             252
       000170     01 CARD-REC   PC X(80).                                         USERS             660
       000180     FD TAPE-OUT                                                     USERS             660
       000190         RECORDING MODE IS NON-STANDARD                              USERS             820
       000200         VALUE OF ID IS "OUTPUT".                                    USERS             820
       000210     01 LABEL.                                                       USERS            1648
       000220         03  FILLER  PC X(9).                                        USERS            1648
       000230         03  MULTI-ID   PC X(6).                                     USERS            1666
       000240         03  FILLER  PC XX.                                          USERS            1678
       000250         03  FILE-ID  PC X(6).                                       USERS            1682
       000260         03  FILLER  PC X.                                           USERS            1694
       000270         03  REEL-NO   PC 999.                                       USERS            1696
       000280         03  DATE-WRITTEN-1  PC 9(5).                                USERS            1702
       000290         03  CYCLE-1  PC 99.                                         USERS            1712
       000300         03  PURGE-DATE  PC 9(5).                                    USERS            1716
       000310         03  FILLER  PC X(41).                                       USERS            1726
       000320         03  USERS-RECORD-COUNT  PC 9(5).                            USERS            1808
       000330         03  USERS-PHYSICAL-REEL  PC 9(5).                           USERS            1818
       000340     01  TAPE-OUT-REC  PC X(200).                                    USERS            1828
       000350     PROCEDURE DIVISION.                                             USERS            1648
       000360     DECLARATIVES.                                                   USERS            1962
       000370     OUTPUT-LABEL SECTION.                                           USERS            1970
       000380         USE AFTER STANDARD BEGINNING FILE LABEL PROCEDURE ON        USERS            1970
       000390           TAPE-OUT.                                                 USERS            1970
       000400     PAR-1.                                                          USERS            2080
       000410         MOVE 1 TU REEL-NU.                                          USERS            2080
       000420         MOVE 68263 TO DATE-WRITTEN-1.                               USERS            2098
       000430         MOVE 15 TO CYCLE-1.                                         USERS            2116
       000440         MOVE 69262 TU PURGE-DATE.                                   USERS            2134
       000450         MOVE 1095 TO USERS-RECORD-COUNT.                            USERS            2152
       000460         MOVE 5000 TO USERS-PHYSICAL-REEL.                           USERS            2170
       000470     END DECLARATIVES.                                               USERS            2188
       000480     BEGIN SECTION.                                                  USERS            2426
       000490     START.                                                          USERS            2426
       000500         OPEN INPUT CARD-IN   OUTPUT TAPE-OUT.                       USERS            2426
       000510     REED.                                                           USERS            2470
       000520         READ CARU-IN AT END GO TO EOJ.                              USERS            2470
       000530         WRITE TAPE-OUT-REC FROM CARD-REC.                           USERS            2504
       000540         GO TO REED.                                                 USERS            2588
       000550     EOJ.                                                            USERS            2596
       000560         CLOSE CARD-IN WITH RELEASE.                                 USERS            2596
       000570         CLOSE TAPE-OUT WITH LOCK.                                   USERS            2618
       000580         STOP RUN.                                                   USERS            2640
       000590     END-UF-JOB.                                                     USERS            2646
COMPILE DATE 10:51  09/27/68  USING 242/68 COMPILER.  PROGRAM ID IS USERS .
ELAPSED TIME   93 SECONDS.  CUMPILER SIZE  27000 HYTES.
ELAPSED TIME IS TOTAL CLOCK TIME, NOT TIME CHARGEABLE TO COMPILATION.

PROGRAM REQUIRES   18 DISK SEGMENTS OF 100 BYTES EACH.
00059 SYMBOLIC RECORDS COMPILED AT   38 RECORDS PER MINUTE.
TOTAL CORE REQUIRED IS   1424 BYTES.
                                            LOW ADDRESS    HIGH ADDRESS    LENGTH IN DIGITS
RESERVED MEMORY                               000000          000252           000252
DATA DIVISION                                 000252          001852           001600
FIXED SEGMENT CONSTANTS                       001852          001944           000092
FIXED SEGMENT INSTRUCTIONS                    001944          002648           000704
STACK                                         002648          003000           000352
```

Figure G-2. Example of Use Procedures to Change Label Fields



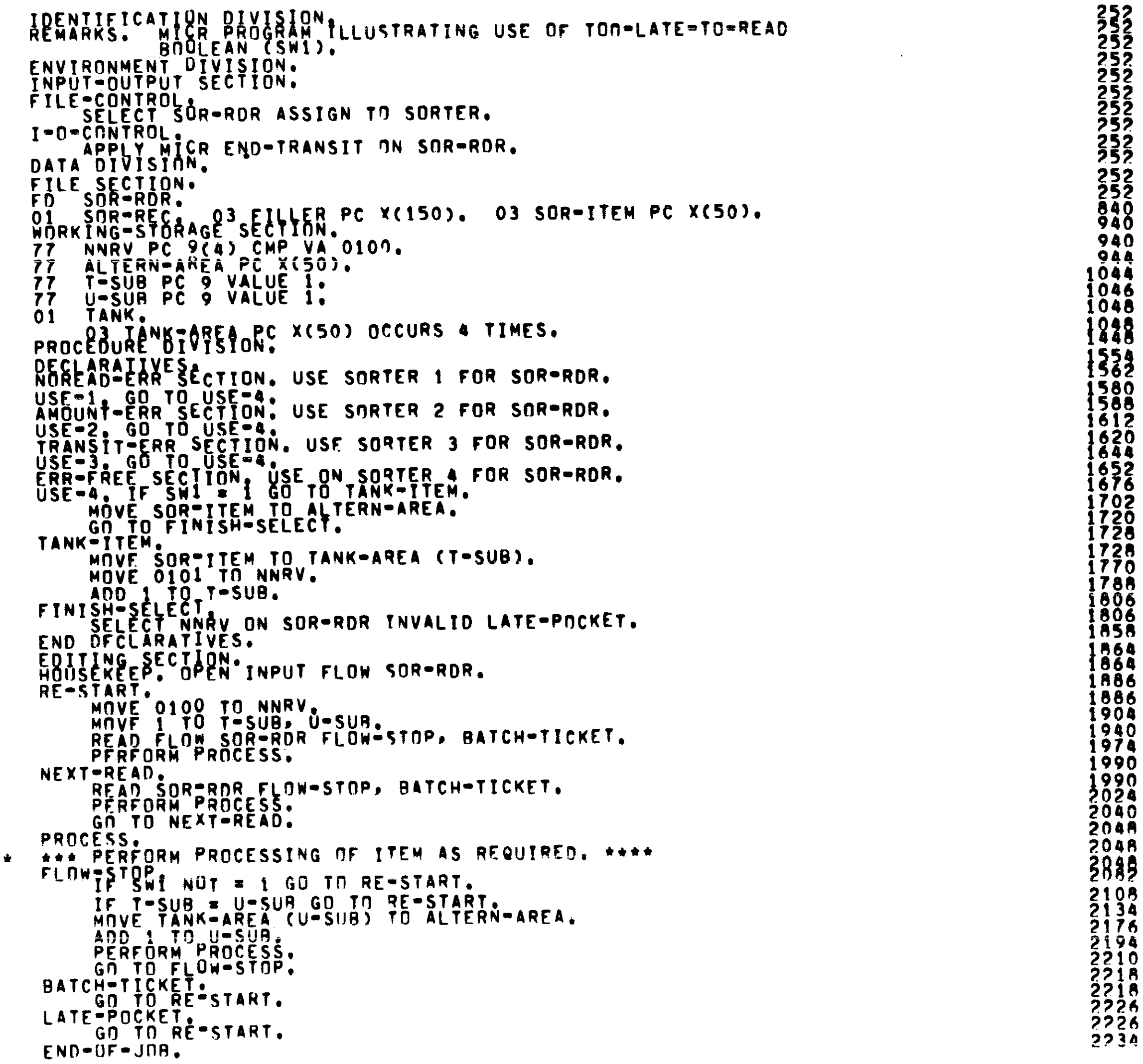
```
     001010     IDENTIFICATION DIVISION.
     001020     ENVIRONMENT DIVISION.                                                                    268
     001030     FILE-CONTROL. SELECT INTEGERS-TO-BE-SQUARED, ASSIGN TO READER.                           268
     001040                   SELECT SUMMARY-PRINT, ASSIGN TO PRINTER.                                   268
     001050     DATA DIVISION.                                                                           268
     001060     FILE SECTION.                                                                            268
     001070     FD INTEGERS-TO-BE-SQUARED.                                                               268
     001080     01  INTEGER-IN                   PC X(13).                                               268
     001090     FD SUMMARY-PRINT.                                                                        676
     001100     01   OUT.                                                                                704
     001110          03 SQROOTS                  PC Z(7).9(5).                                          1112
     001120          03 FILLER                   PC X(4).                                               1112
     001130          03 INPUT-INTEGER            PC Z(13).9(5).                                         1138
     001140          03 FILLER                   PC X(4).                                               1146
     001150          03 PROOF                    PC Z(13).9(5).                                         1184
     001160          03 FILLER                   PC X(75) VALUE SPACES.                                 1192
     001170     WORKING-STORAGE SECTION.                                                                1230
     001180          77 HOLD-3                   PC 9(13)V9(5).                                         1380
     001190          77 X                        PC 9(13)V9(5).                                         1380
     001200          77 Y                        PC 9(13)V9(5).                                         1416
     001210          77 YO                       PC 9(13)V9(5).                                         1452
     001220          77 ROOT-STGE-OUT            PC 9(7)V9(5).                                          1488
     001230     PROCEDURE DIVISION.                                                                     1524
     001240     SQUARE-ROOT SECTION.                                                                    1548
     001250     INIT. OPEN INPUT INTEGERS-TO-BE-SQUARED, OUTPUT SUMMARY-PRINT.                          1658
     001251          MOVE SPACES TO OUT.                                                                1658
     001260     READ-A-RECORD.                                                                          1702
     001270          WRITE OUT BEFORE ADVANCING 02 LINES.                                               1738
     001280          READ INTEGERS-TO-BE-SQUARED, AT END GO TO FINI.                                    1738
     001281     COPYGRAPH. COPY "SQROOT" REPLACING AAA BY INTEGER-IN,                                   1768
     001282          BBB BY INPUT-INTEGER,                                                              1802
     001290          CCC BY X,  DDD BY YO, EEE BY HOLD-3,FFF BY Y,                                      1802
     001300          GGG BY ROOT-STGE-OUT, HHH BY SQROOTS, III BY PROOF,                                1802
     001310          123 BY READ-A-RECORD.                                                              1802
    L001020 *        CREATES THE SQUARE ROOT OF INTEGERS UP TO 13 DIGITS IN LENGTH                      1802
    L001030 *        GIVING THE MAXIMUM SIZE RESULT OF 7 DIGITS IN FRONT OF THE                         1802
    L001040 *        DECIMAL AND 5 AFTER. ALL INPUT INTEGERS MUST BE RIGHT JUSTIFD                      1802
    L001050 *        WITHIN THE FIELD. THE FOLLOWING FIELDS MUST BE SUPPLIED BY A                       1802
    L001060 *        COPY SQROOT REPLACING CLAUSE,                                                      1802
    L001070 *        AAA = INPUT DATA FIELD,            CAN BE UP TO A PC X(13)                         1802
    L001080 *        BBB = OUTPUT PRINT AREA TO REFLECT ORIGINAL DATA,  Z(13).9(5)                      1802
    L001090 *        CCC = WORK AREA #1,                77 LEVEL ENTRY  PC 9(13)V9(5)                   1802
    L001100 *        DDD = WORK AREA #2,                77 LEVEL ENTRY  PC 9(13)V9(5)                   1802
    L001110 *        EEE = WORK AREA #3,                77 LEVEL ENTRY  PC 9(13)V9(5)                   1802
    L001120 *        FFF = WORK AREA #4,                77 LEVEL ENTRY  PC 9(13)V9(5)                   1802
    L001130 *        GGG = SQ-ROOT PROOF AREA,          77 LEVEL ENTRY  PC 9(7)V9(5)                    1802
    L001140 *        HHH = SQUARE ROOT RESULT IN PRINT AREA,            PC Z(7).9(5)                    1802
    L001150 *        III = PROOF RESULT IN OUTPUT PRINT AREA,           PC Z(13).9(4)                   1802
    L001160 *        123 = EXIT TO PROGRAM AFTER SQUARE ROOT HAS BEEN CREATED.                          1802
    L001170 *   INTEGER-CHECK.                                                                          1802
    L001180          EXAMINE AAA REPLACING ALL SPACES BY ZERO.                                          1802
    L001190          MOVE AAA TO BBB, CCC, DDD.                                                         1802
    L001200     FORMULA-ROOT-ENTRY.                                                                     1890
    L001210          DIVIDE DDD INTO CCC GIVING EEE.                                                    2022
    L001220          ADD DDD TO EEE.                                                                    2022
    L001230          MULTIPLY EEE BY .5 GIVING FFF.                                                     2100
    L001240           IF FFF = DDD, MOVE FFF TO GGG, HHH,                                               2118
    L001250          MULTIPLY GGG BY GGG GIVING III, GO TO 123.                                         2160
    L001260          MOVE FFF TO DDD, GO TO FORMULA-ROOT-ENTRY.                                         2204
     001320     FINI. STOP RUN.                                                                         2276
     999998     END-OF-JOB.                                                                             2310
COMPILE DATE 08:06  12/07/68  USING 008/69 COMPILER, PROGRAM ID IS ??????.                              2316
ELAPSED TIME   79 SECONDS.  COMPILER SIZE  17000 BYTES.
ELAPSED TIME IS TOTAL CLOCK TIME, NOT TIME CHARGEABLE TO COMPILATION.
PROGRAM REQUIRES   17 DISK SEGMENTS OF 100 BYTES EACH.
00061 SYMBOLIC RECORDS COMPILED AT   46 RECORDS PER MINUTE.
TOTAL CORE REQUIRED IS   1178 BYTES.
TOTAL NUMBER OF WARNING MESSAGES IS  003 INCLUDING  NO SEQUENCE ERRORS.

                                      LOW ADDRESS    HIGH ADDRESS    LENGTH IN DIGITS
RESERVED MEMORY                          000000          000252           000252
DATA DIVISION                            000252          001636           001384
FIXED SEGMENT CONSTANTS                  001636          001640           000004
FIXED SEGMENT INSTRUCTIONS               001640          002316           000676
STACK                                    002316          003000           000684
```

Figure G-3. Example of Copy Replacing and Square Root Program



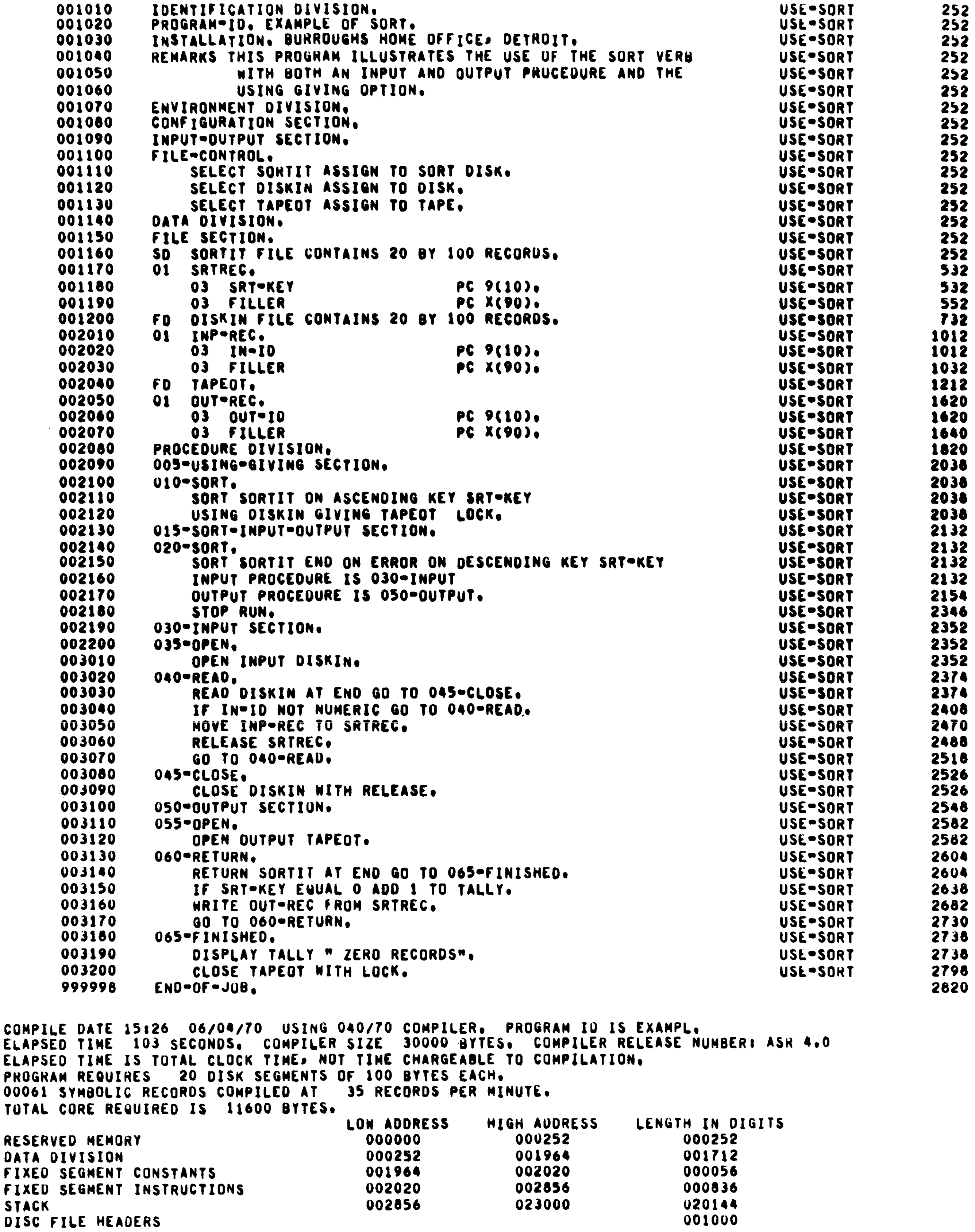
```
Extracted Square          Input       Square of Extracted Square
     1.00000             1.00000               1.00000
     1.41421             2.00000               1.99998
     1.73205             3.00000               2.99999
     2.00000             4.00000               4.00000
     2.23606             5.00000               4.99996
     2.23606             5.00000               4.99996
     2.44948             6.00000               5.99995
     2.64575             7.00000               6.99999
     2.64575             7.00000               6.99999
     5.00000            25.00000              25.00000
     5.47722            30.00000              29.99993
     6.32455            40.00000              39.99993
     7.07106            50.00000              49.99988
```

Figure G-4. Example Output of Copy Replacing and Square Root Program



```
 IDENTIFICATION DIVISION.                                              252
 REMARKS.  MICR PROGRAM ILLUSTRATING USE OF TOO-LATE-TO-READ           252
           BOOLEAN (SW1).                                              252
 ENVIRONMENT DIVISION.                                                 252
 INPUT-OUTPUT SECTION.                                                 252
 FILE-CONTROL.                                                         252
     SELECT SOR-RDR ASSIGN TO SORTER.                                  252
 I-O-CONTROL.                                                          252
     APPLY MICR END-TRANSIT ON SOR-RDR.                                252
 DATA DIVISION.                                                        252
 FILE SECTION.                                                         252
 FD  SOR-RDR.                                                          252
 01  SOR-REC.  03 FILLER PC X(150).  03 SOR-ITEM PC X(50).             840
 WORKING-STORAGE SECTION.                                              940
 77  NNRV PC 9(4) CMP VA 0100.                                         940
 77  ALTERN-AREA PC X(50).                                             944
 77  T-SUB PC 9 VALUE 1.                                              1044
 77  U-SUB PC 9 VALUE 1.                                              1046
 01  TANK.                                                            1048
     03 TANK-AREA PC X(50) OCCURS 4 TIMES.                            1048
 PROCEDURE DIVISION.                                                  1448
 DECLARATIVES.                                                        1554
 NOREAD-ERR SECTION. USE SORTER 1 FOR SOR-RDR.                        1562
 USE-1. GO TO USE-4.                                                  1580
 AMOUNT-ERR SECTION. USE SORTER 2 FOR SOR-RDR.                        1588
 USE-2. GO TO USE-4.                                                  1612
 TRANSIT-ERR SECTION. USE SORTER 3 FOR SOR-RDR.                       1620
 USE-3. GO TO USE-4.                                                  1644
 ERR-FREE SECTION. USE ON SORTER 4 FOR SOR-RDR.                       1652
 USE-4. IF SW1 = 1 GO TO TANK-ITEM.                                   1676
     MOVE SOR-ITEM TO ALTERN-AREA.                                    1702
     GO TO FINISH-SELECT.                                             1720
 TANK-ITEM.                                                           1728
     MOVE SOR-ITEM TO TANK-AREA (T-SUB).                              1728
     MOVE 0101 TO NNRV.                                               1770
     ADD 1 TO T-SUB.                                                  1788
 FINISH-SELECT.                                                       1806
     SELECT NNRV ON SOR-RDR INVALID LATE-POCKET.                      1806
 END DECLARATIVES.                                                    1858
 EDITING SECTION.                                                     1864
 HOUSEKEEP. OPEN INPUT FLOW SOR-RDR.                                  1864
 RE-START.                                                            1886
     MOVE 0100 TO NNRV.                                               1886
     MOVE 1 TO T-SUB, U-SUB.                                          1904
     READ FLOW SOR-RDR FLOW-STOP, BATCH-TICKET.                       1940
     PERFORM PROCESS.                                                 1974
 NEXT-READ.                                                           1990
     READ SOR-RDR FLOW-STOP, BATCH-TICKET.                            1990
     PERFORM PROCESS.                                                 2024
     GO TO NEXT-READ.                                                 2040
 PROCESS.                                                             2048
*  *** PERFORM PROCESSING OF ITEM AS REQUIRED. ****                   2048
 FLOW-STOP.                                                           2048
     IF SW1 NOT = 1 GO TO RE-START.                                   2082
     IF T-SUB = U-SUB GO TO RE-START.                                 2108
     MOVE TANK-AREA (U-SUB) TO ALTERN-AREA.                           2134
     ADD 1 TO U-SUB.                                                  2176
     PERFORM PROCESS.                                                 2194
     GO TO FLOW-STOP.                                                 2210
 BATCH-TICKET.                                                        2218
     GO TO RE-START.                                                  2218
 LATE-POCKET.                                                         2226
     GO TO RE-START.                                                  2226
 END-OF-JOB.                                                          2234
```

Figure G-5. Example of MICR Reader Sorter

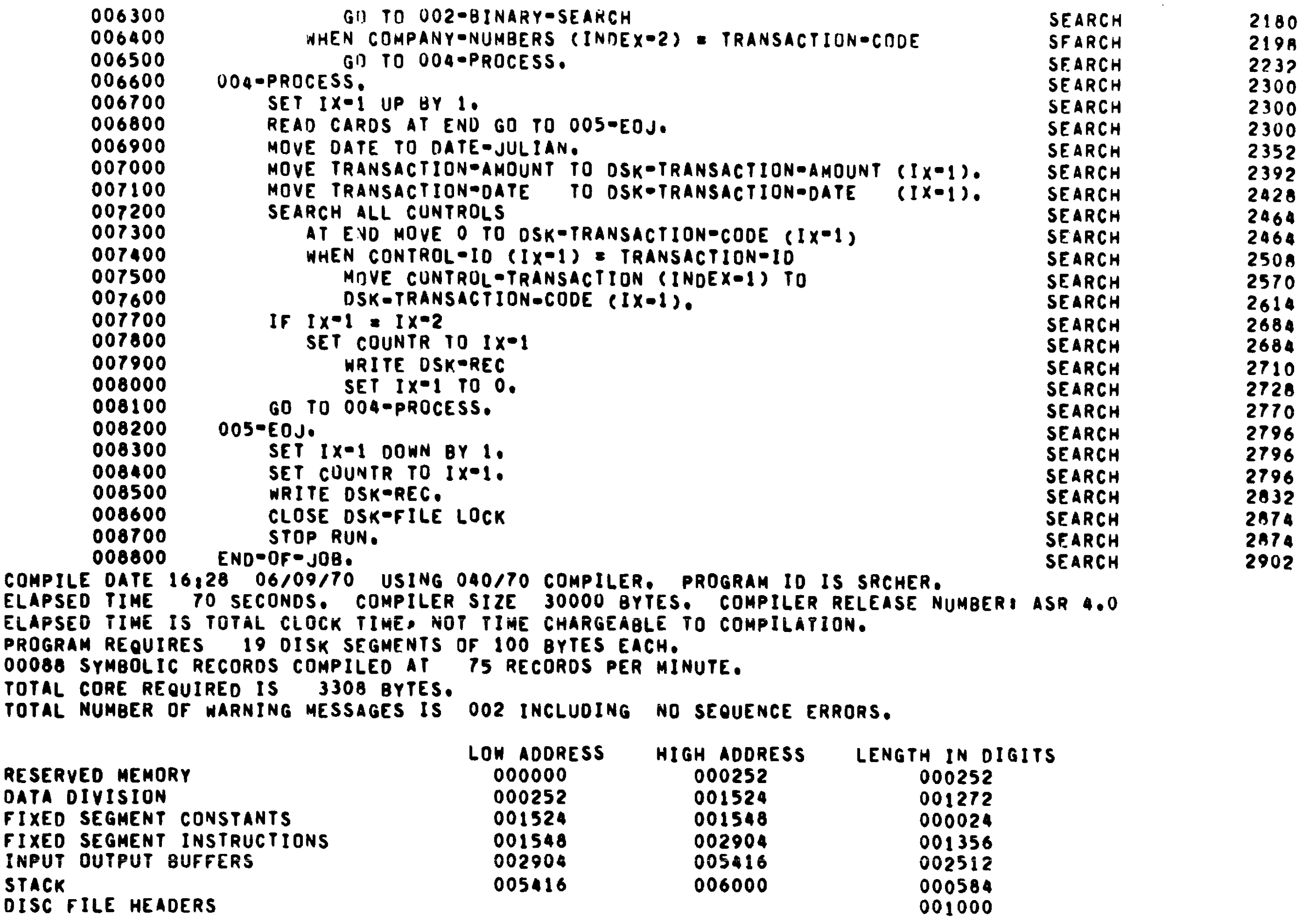


```
001010     IDENTIFICATION DIVISION.                                          USE-SORT       252
001020     PROGRAM-ID. EXAMPLE OF SORT.                                      USE-SORT       252
001030     INSTALLATION. BURROUGHS HOME OFFICE, DETROIT.                     USE-SORT       252
001040     REMARKS THIS PROGRAM ILLUSTRATES THE USE OF THE SORT VERB         USE-SORT       252
001050             WITH BOTH AN INPUT AND OUTPUT PROCEDURE AND THE           USE-SORT       252
001060             USING GIVING OPTION.                                      USE-SORT       252
001070     ENVIRONMENT DIVISION.                                             USE-SORT       252
001080     CONFIGURATION SECTION.                                            USE-SORT       252
001090     INPUT-OUTPUT SECTION.                                             USE-SORT       252
001100     FILE-CONTROL.                                                     USE-SORT       252
001110         SELECT SORTIT ASSIGN TO SORT DISK.                            USE-SORT       252
001120         SELECT DISKIN ASSIGN TO DISK.                                 USE-SORT       252
001130         SELECT TAPEOT ASSIGN TO TAPE.                                 USE-SORT       252
001140     DATA DIVISION.                                                    USE-SORT       252
001150     FILE SECTION.                                                     USE-SORT       252
001160     SD  SORTIT FILE CONTAINS 20 BY 100 RECORDS.                       USE-SORT       252
001170     01  SRTREC.                                                       USE-SORT       532
001180         03  SRT-KEY                 PC 9(10).                         USE-SORT       532
001190         03  FILLER                  PC X(90).                         USE-SORT       552
001200     FD  DISKIN FILE CONTAINS 20 BY 100 RECORDS.                       USE-SORT       732
002010     01  INP-REC.                                                      USE-SORT      1012
002020         03  IN-ID                   PC 9(10).                         USE-SORT      1012
002030         03  FILLER                  PC X(90).                         USE-SORT      1032
002040     FD  TAPEOT.                                                       USE-SORT      1212
002050     01  OUT-REC.                                                      USE-SORT      1620
002060         03  OUT-ID                  PC 9(10).                         USE-SORT      1620
002070         03  FILLER                  PC X(90).                         USE-SORT      1640
002080     PROCEDURE DIVISION.                                               USE-SORT      1820
002090     005-USING-GIVING SECTION.                                         USE-SORT      2038
002100     010-SORT.                                                         USE-SORT      2038
002110         SORT SORTIT ON ASCENDING KEY SRT-KEY                          USE-SORT      2038
002120         USING DISKIN GIVING TAPEOT  LOCK.                             USE-SORT      2038
002130     015-SORT-INPUT-OUTPUT SECTION.                                    USE-SORT      2132
002140     020-SORT.                                                         USE-SORT      2132
002150         SORT SORTIT END ON ERROR ON DESCENDING KEY SRT-KEY            USE-SORT      2132
002160         INPUT PROCEDURE IS 030-INPUT                                  USE-SORT      2132
002170         OUTPUT PROCEDURE IS 050-OUTPUT.                               USE-SORT      2154
002180         STOP RUN.                                                     USE-SORT      2346
002190     030-INPUT SECTION.                                                USE-SORT      2352
002200     035-OPEN.                                                         USE-SORT      2352
003010         OPEN INPUT DISKIN.                                            USE-SORT      2352
003020     040-READ.                                                         USE-SORT      2374
003030         READ DISKIN AT END GO TO 045-CLOSE.                           USE-SORT      2374
003040         IF IN-ID NOT NUMERIC GO TO 040-READ.                          USE-SORT      2408
003050         MOVE INP-REC TO SRTREC.                                       USE-SORT      2470
003060         RELEASE SRTREC.                                               USE-SORT      2488
003070         GO TO 040-READ.                                               USE-SORT      2518
003080     045-CLOSE.                                                        USE-SORT      2526
003090         CLOSE DISKIN WITH RELEASE.                                    USE-SORT      2526
003100     050-OUTPUT SECTION.                                               USE-SORT      2548
003110     055-OPEN.                                                         USE-SORT      2582
003120         OPEN OUTPUT TAPEOT.                                           USE-SORT      2582
003130     060-RETURN.                                                       USE-SORT      2604
003140         RETURN SORTIT AT END GO TO 065-FINISHED.                      USE-SORT      2604
003150         IF SRT-KEY EQUAL 0 ADD 1 TO TALLY.                            USE-SORT      2638
003160         WRITE OUT-REC FROM SRTREC.                                    USE-SORT      2682
003170         GO TO 060-RETURN.                                             USE-SORT      2730
003180     065-FINISHED.                                                     USE-SORT      2738
003190         DISPLAY TALLY " ZERO RECORDS".                                USE-SORT      2738
003200         CLOSE TAPEOT WITH LOCK.                                       USE-SORT      2798
999998     END-OF-JOB.                                                                     2820

COMPILE DATE 15:26  06/04/70  USING 040/70 COMPILER.  PROGRAM ID IS EXAMPL.
ELAPSED TIME  103 SECONDS.  COMPILER SIZE  30000 BYTES.  COMPILER RELEASE NUMBER: ASR 4.0
ELAPSED TIME IS TOTAL CLOCK TIME, NOT TIME CHARGEABLE TO COMPILATION.
PROGRAM REQUIRES   20 DISK SEGMENTS OF 100 BYTES EACH.
00061 SYMBOLIC RECORDS COMPILED AT    35 RECORDS PER MINUTE.
TOTAL CORE REQUIRED IS  11600 BYTES.
                                    LOW ADDRESS    HIGH ADDRESS    LENGTH IN DIGITS
RESERVED MEMORY                       000000          000252           000252
DATA DIVISION                         000252          001964           001712
FIXED SEGMENT CONSTANTS               001964          002020           000056
FIXED SEGMENT INSTRUCTIONS            002020          002856           000836
STACK                                 002856          023000           020144
DISC FILE HEADERS                                                      001000
```

Figure G-6. Example of Sort Program

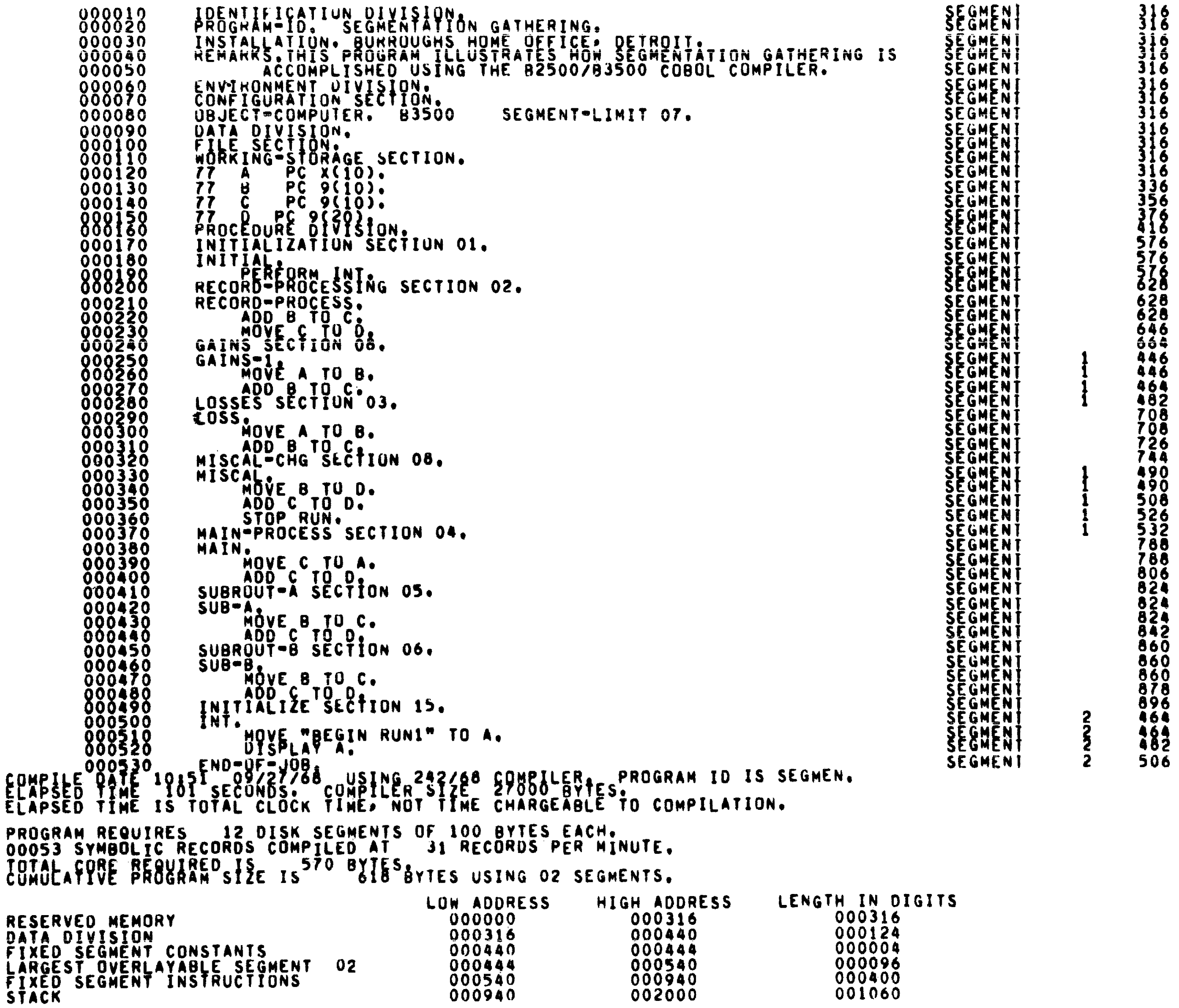
```
IDENTIFICATION DIVISION.                                          SEARCH           252
PROGRAM-ID.  SRCHER.                                              SEARCH           252
REMARKS. EXAMPLE OF SEARCH VERB USING OPTIONS 1 AND 2.            SEARCH           252
ENVIRONMENT DIVISION.                                             SEARCH           252
CONFIGURATION SECTION.                                            SEARCH           252
INPUT-OUTPUT SECTION.                                             SEARCH           252
FILE-CONTROL.                                                     SEARCH           252
    SELECT CARDS ASSIGN TO READER RESERVE 2.                      SEARCH           252
    SELECT DSK-FILE ASSIGN TO DISK ACCESS SEQUENTIAL.             SEARCH           252
DATA DIVISION.                                                    SEARCH           252
FILE SECTION.                                                     SEARCH           252
FD  CARDS.                                                        SEARCH           252
01  CARD-REC.                                                     SEARCH           740
    03  TRANSACTION-ID                  PIC X(6).                 SEARCH           740
    03  TRANSACTION-SPEC REDEFINES TRANSACTION-ID.                SEARCH           752
        05  TRANSACTION-CODE            PIC 99.                   SEARCH           740
        05  FILLER                      PIC X(4).                 SEARCH           744
    03  TRANSACTION-DATE                PIC 9(6).                 SEARCH           752
    03  TRANSACTION-AMOUNT              PIC 9(8)V99.              SEARCH           764
    03  FILLER                          PIC X(58).                SEARCH           784
FD  DSK-FILE                                                      SEARCH           784
    FILE CONTAINS 20 BY 500 RECORDS                               SEARCH           900
    VALUE OF IDENTIFICATION "TRAN12"                              SEARCH           900
    BLOCK CONTAINS 10 RECORDS.                                    SEARCH           900
01  DSK-REC.                                                      SEARCH          1180
    03  DATE-JULIAN                     PIC 999     CMP.          SEARCH          1180
    03  COUNTR                      USAGE IS INDEX.               SEARCH          1183
    03  DSK-ARRAY   OCCURS 8  INDEXED BY IX-1  IX-2.              SEARCH          1188
        05  DSK-TRANSACTION-CODE        PIC 99      CMP.          SEARCH          1188
        05  DSK-TRANSACTION-DATE        PIC 9(6).                 SEARCH          1190
        05  DSK-TRANSACTION-AMOUNT      PIC 9(8)V99 CMP.          SEARCH          1202
WORKING-STORAGE SECTION.                                          SEARCH          1380
77  TEST-VALUE     VALUE 6              PIC 9(6)    CMP.          SEARCH          1380
01  CONTROL-VALUES.                                               SEARCH          1386
    03  FILLER     VALUE "A12345"       PIC X(6).                 SEARCH          1388
    03  FILLER     VALUE 01             PIC 99      CMP.          SEARCH          1400
    03  FILLER     VALUE "B12345"       PIC X(6).                 SEARCH          1402
    03  FILLER     VALUE 02             PIC 99      CMP.          SEARCH          1414
01  CONTROL-ARRAY REDEFINES CONTROL-VALUES.                       SEARCH          1416
    03  CONTROLS    OCCURS 2  INDEXED BY INDEX-1.                 SEARCH          1388
        05  CONTROL-ID                  PIC X(6).                 SEARCH          1388
        05  CONTROL-TRANSACTION         PIC 99      CMP.          SEARCH          1400
01  COMPANY-VALUE.                                                SEARCH          1402
    03  FILLER                          PIC 9(24)   CMP           SEARCH          1416
                   VALUE 010203040506070809101112.                SEARCH          1416
    03  FILLER                          PIC 9(24)   CMP           SEARCH          1440
                   VALUE 131415161718192021222324.                SEARCH          1440
01  COMPANY-TABLE REDEFINES COMPANY-VALUE.                        SEARCH          1464
    03  COMPANY-NUMBERS                 PIC 99      CMP           SEARCH          1416
                    OCCURS 24 INDEXED BY INDEX-2.                 SEARCH          1416
PROCEDURE DIVISION.                                               SEARCH          1464
001-START.                                                        SEARCH          1852
    OPEN INPUT CARDS.                                             SEARCH          1852
    OPEN OUTPUT DSK-FILE.                                         SEARCH          1874
    SET IX-1 TO 0     SET IX-2 TO 8     MOVE 6 TO TALLY.          SEARCH          1896
    READ CARDS.                                                   SEARCH          1950
002-BINARY-SEARCH.                                                SEARCH          1976
    SET INDEX-2 TO TALLY.                                         SEARCH          1976
    SEARCH COMPANY-NUMBERS VARYING INDEX-2     AT END STOP RUN    SEARCH          2036
       WHEN COMPANY-NUMBERS   (INDEX-2) > TRANSACTION-CODE        SEARCH          2062
          COMPUTE TALLY = TEST-VALUE / 2                          SEARCH          2094
          MOVE TALLY TO TEST-VALUE                                SEARCH          2120
```

Figure G-7. Example of SEARCH Verb Usage (Sheet 1 of 2)



```
       006300                GO TO 002-BINARY-SEARCH                                 SEARCH         2180
       006400             WHEN COMPANY-NUMBERS (INDEX-2) = TRANSACTION-CODE          SFARCH         2198
       006500                GO TO 004-PROCESS.                                      SEARCH         2232
       006600       004-PROCESS.                                                     SEARCH         2300
       006700           SET IX-1 UP BY 1.                                            SEARCH         2300
       006800           READ CARDS AT END GO TO 005-EOJ.                             SEARCH         2300
       006900           MOVE DATE TO DATE-JULIAN.                                    SEARCH         2352
       007000           MOVE TRANSACTION-AMOUNT TO DSK-TRANSACTION-AMOUNT (IX-1).    SEARCH         2392
       007100           MOVE TRANSACTION-DATE   TO DSK-TRANSACTION-DATE   (IX-1).    SEARCH         2428
       007200           SEARCH ALL CUNTROLS                                          SEARCH         2464
       007300              AT END MOVE 0 TO DSK-TRANSACTION-CODE (IX-1)              SEARCH         2464
       007400              WHEN CONTROL-ID (IX-1) = TRANSACTION-ID                   SEARCH         2508
       007500                 MOVE CONTROL-TRANSACTION (INDEX-1) TO                  SEARCH         2570
       007600                 DSK-TRANSACTION-CODE (IX-1).                           SEARCH         2614
       007700           IF IX-1 = IX-2                                               SEARCH         2684
       007800              SET COUNTR TO IX-1                                        SEARCH         2684
       007900                 WRITE DSK-REC                                          SEARCH         2710
       008000                 SET IX-1 TO 0.                                         SEARCH         2728
       008100           GO TO 004-PROCESS.                                           SEARCH         2770
       008200       005-EOJ.                                                         SEARCH         2796
       008300           SET IX-1 DOWN BY 1.                                          SEARCH         2796
       008400           SET COUNTR TO IX-1.                                          SEARCH         2796
       008500           WRITE DSK-REC.                                               SEARCH         2832
       008600           CLOSE DSK-FILE LOCK                                          SEARCH         2874
       008700           STOP RUN.                                                    SEARCH         2874
       008800       END-OF-JOB.                                                      SEARCH         2902
COMPILE DATE 16:28  06/09/70  USING 040/70 COMPILER.  PROGRAM ID IS SRCHER.
ELAPSED TIME   70 SECONDS.  COMPILER SIZE  30000 BYTES.  COMPILER RELEASE NUMBER: ASR 4.0
ELAPSED TIME IS TOTAL CLOCK TIME, NOT TIME CHARGEABLE TO COMPILATION.
PROGRAM REQUIRES   19 DISK SEGMENTS OF 100 BYTES EACH.
00088 SYMBOLIC RECORDS COMPILED AT   75 RECORDS PER MINUTE.
TOTAL CORE REQUIRED IS   3308 BYTES.
TOTAL NUMBER OF WARNING MESSAGES IS  002 INCLUDING  NO SEQUENCE ERRORS.

                                    LOW ADDRESS    HIGH ADDRESS    LENGTH IN DIGITS
RESERVED MEMORY                       000000          000252           000252
DATA DIVISION                         000252          001524           001272
FIXED SEGMENT CONSTANTS               001524          001548           000024
FIXED SEGMENT INSTRUCTIONS            001548          002904           001356
INPUT OUTPUT BUFFERS                  002904          005416           002512
STACK                                 005416          006000           000584
DISC FILE HEADERS                                                      001000
```

Figure G-7. Example of SEARCH Verb Usage (Sheet 2 of 2)



```
000010     IDENTIFICATION DIVISION.                                          SEGMENT        316
000020     PROGRAM-ID.  SEGMENTATION GATHERING.                              SEGMENT        316
000030     INSTALLATION. BURROUGHS HOME OFFICE, DETROIT.                     SEGMENT        316
000040     REMARKS.THIS PROGRAM ILLUSTRATES HOW SEGMENTATION GATHERING IS    SEGMENT        316
000050              ACCOMPLISHED USING THE B2500/B3500 COBOL COMPILER.       SEGMENT        316
000060     ENVIRONMENT DIVISION.                                             SEGMENT        316
000070     CONFIGURATION SECTION.                                            SEGMENT        316
000080     OBJECT-COMPUTER.  B3500    SEGMENT-LIMIT 07.                      SEGMENT        316
000090     DATA DIVISION.                                                    SEGMENT        316
000100     FILE SECTION.                                                     SEGMENT        316
000110     WORKING-STORAGE SECTION.                                          SEGMENT        316
000120     77  A    PC X(10).                                                SEGMENT        316
000130     77  B    PC 9(10).                                                SEGMENT        336
000140     77  C    PC 9(10).                                                SEGMENT        356
000150     77  D  PC 9(20).                                                  SEGMENT        376
000160     PROCEDURE DIVISION.                                               SEGMENT        416
000170     INITIALIZATION SECTION 01.                                        SEGMENT        576
000180     INITIAL.                                                          SEGMENT        576
000190          PERFORM INT.                                                 SEGMENT        576
000200     RECORD-PROCESSING SECTION 02.                                     SEGMENT        628
000210     RECORD-PROCESS.                                                   SEGMENT        628
000220          ADD B TO C.                                                  SEGMENT        628
000230          MOVE C TO D.                                                 SEGMENT        646
000240     GAINS SECTION 08.                                                 SEGMENT        664
000250     GAINS-1.                                                          SEGMENT    1   446
000260          MOVE A TO B.                                                 SEGMENT    1   446
000270          ADD B TO C.                                                  SEGMENT    1   464
000280     LOSSES SECTION 03.                                                SEGMENT    1   482
000290     LOSS.                                                             SEGMENT        708
000300          MOVE A TO B.                                                 SEGMENT        708
000310          ADD B TO C.                                                  SEGMENT        726
000320     MISCAL-CHG SECTION 08.                                            SEGMENT        744
000330     MISCAL.                                                           SEGMENT    1   490
000340          MOVE B TO D.                                                 SEGMENT    1   490
000350          ADD C TO D.                                                  SEGMENT    1   508
000360          STOP RUN.                                                    SEGMENT    1   526
000370     MAIN-PROCESS SECTION 04.                                          SEGMENT    1   532
000380     MAIN.                                                             SEGMENT        788
000390          MOVE C TO A.                                                 SEGMENT        788
000400          ADD C TO D.                                                  SEGMENT        806
000410     SUBROUT-A SECTION 05.                                             SEGMENT        824
000420     SUB-A.                                                            SEGMENT        824
000430          MOVE B TO C.                                                 SEGMENT        824
000440          ADD C TO D.                                                  SEGMENT        842
000450     SUBROUT-B SECTION 06.                                             SEGMENT        860
000460     SUB-B.                                                            SEGMENT        860
000470          MOVE B TO C.                                                 SEGMENT        860
000480          ADD C TO D.                                                  SEGMENT        878
000490     INITIALIZE SECTION 15.                                            SEGMENT        896
000500     INT.                                                              SEGMENT    2   464
000510          MOVE "BEGIN RUN1" TO A.                                      SEGMENT    2   464
000520          DISPLAY A.                                                   SEGMENT    2   482
000530     END-OF-JOB.                                                       SEGMENT    2   506
COMPILE DATE 10:51  09/27/68  USING 242/68 COMPILER.  PROGRAM ID IS SEGMEN.
ELAPSED TIME  101 SECONDS.  COMPILER SIZE  27000 BYTES.
ELAPSED TIME IS TOTAL CLOCK TIME, NOT TIME CHARGEABLE TO COMPILATION.

PROGRAM REQUIRES   12 DISK SEGMENTS OF 100 BYTES EACH.
00053 SYMBOLIC RECORDS COMPILED AT   31 RECORDS PER MINUTE.
TOTAL CORE REQUIRED IS    570 BYTES.
CUMULATIVE PROGRAM SIZE IS     618 BYTES USING 02 SEGMENTS.

                                         LOW ADDRESS    HIGH ADDRESS   LENGTH IN DIGITS
RESERVED MEMORY                            000000          000316          000316
DATA DIVISION                              000316          000440          000124
FIXED SEGMENT CONSTANTS                    000440          000444          000004
LARGEST OVERLAYABLE SEGMENT  02            000444          000540          000096
FIXED SEGMENT INSTRUCTIONS                 000540          000940          000400
STACK                                      000940          002000          001060
```

Figure G-8. Example of Segmentation Gathering

Sections 1, 2, 3, 4, 5, and 6, will be gathered into one fixed non-overlayable segment. The memory map at the end of the COBOL listing specifies the amount of memory required for "Fixed Segment Instructions".

Gains Section 08 and Miscal-chg Section 08, will be gathered as the first overlayable segment because it is above the SEGMENT-LIMIT as specified in the OBJECT-COMPUTER paragraph. Initialization Section 15, is the second overlayable segment.

Section 15, is the largest of the two (Section 8 and 15) overlayable segments within the program; therefore, the memory map at the end of the COBOL listing specifies the amount of memory required for "Largest Overlayable Segment 02".

APPENDIX H
UTILIZATION OF CORE DUMP FOR B 2500/B 3500/B 4500 COBOL DEBUGGING

GENERAL.

Frequently the debugging stage of program development requires an examination of object-time-events during the program's execution as a post-mortem evaluation.

The B 2500/B 3500/B 4500 software systems offer several means for such examination. This appendix examines one of them; the program initiated memory "snapshot" (core dump).

A memory dump is frequently the major contributing data available for determining the cause of a DS or DP condition. It can also be a valuable tool in other circumstances, such as examining a trace, monitor, file dump, or, other data.

Before examining the format, content, and/or use of a given memory dump, it will be beneficial to summarize the mechanics by which a memory dump can be obtained.

OBTAINING A MEMORY DUMP.

A memory dump may be initiated in the following ways:

a. By the operator. At any time during program execution, the operator may DM (dump memory and continue) or DP (dump memory and discontinue) the program.

b. By the MCP. If, in conjunction with the EXECUTE Card, a ? MEMDUMP Card is included, then an abnormal EOJ of the program causes an automatic core dump to be initiated.

c. Programmatically. At any point in the program, the statement TRACE 20 (in COBOL) will cause a "snapshot" memory dump to be taken at that point in the program during execution, and is equivalent to a DM, e.g., the program continues after the dump has been accomplished.

CORE DUMP FORMAT.

All core dumps consist of the following particular and general format:

a. Headings and general program information (name, status, Base and Limit registers, address register, etc.).

b. File information blocks (FIB) which consist of 200 digit blocks maintained within the specified program's area and contains flags, counters, and addresses detailing the nature of the file, and their machine absolute address. Also contained is other file information pertaining to all files attached to the program at that given moment, and IOAT flags (input-output Assignment Table)- a MCP table containing flags, links, and addresses detailing the status of the file and the device involved.

c. The contents of memory, between the Base and Limit registers, is printed 100 digits of core locations to a line. The letters A through F represent the hexadecimal undigits 10 through 15 which are composed of consecutive bit combinations 1010 through 1111. Large areas which are greater than 100 digits, and which contain only the digit zero (0) are shown as a single line of asterisks (*); the address on the next line shows the end of the "blank" area. Each print line, prior to the dumped memory information, reflects the program's base relative address, and pertains to the first digit on that line.

COBOL PROGRAM MEMORY ORGANIZATION.

The main body of the core dump can best be understood by a discussion of a COBOL Program's memory organization and structure. The several portions of a given COBOL Program that can be readily distinguished are:

a. Reserved Memory.

1) Program Reserved Memory.

2) COBOL Program reserved areas.

b. FILE SECTION (FIB's, buffer links, label areas, and record areas).

c. WORKING STORAGE entries.

d. Fixed segment constants.

e. Auxiliary code.

f. Overlayable code.

g. Fixed segment code.

h. Buffers.

i. Stack.

j. Disk File Headers and IOAT entries.

With the exception of Program Reserved Memory, none of the above ever occur at fixed locations, and some may not appear at all. A more detailed discussion of these areas is contained in the paragraphs that follow.

PROGRAM RESERVED MEMORY.

Program Reserved Memory (locations 0 through 63) of the program begins at base relative location zero (0), and is of the same format for all B 2500/B 3500/B 4500 Programs. Its format is:

Location	Type	Remarks
0-7	Switches	SW8, SW1 through SW7, one digit each.
8-15	IX1	Seven digits plus a high-order sign and is used in subscripting.
16-23	IX2	Same as IX1 above.

Location	Type	Remarks
24-31	IX3	Points to the current stack entry when PERFORMS (NTR) are active.
32-33		Normally not used.
34-37		Used by block/deblock code.
38-39	Indirect field length counter	Used primarily in EXAMINE constructs.
40-45	Stack pointer	Points to the next available area in the stack for PERFORM (NTR) entries.
46-47		Not used.
48-63	Edit Table	Eight bytes of edit insertion characters.

COBOL RESERVED MEMORY.

COBOL Reserved Memory (locations 63 through x) is variable in length though it frequently terminates at location 252. It contains TALLY, TIME, DATE, program's segment dictionary, and edit macro operator strings.

FILE SECTION.

This section contains the program files (if any) and is structured as follows:

a. File Information Blocks (FIB).
b. Buffer links and descriptors.
c. Label area.
d. Record area (work area).

Structures a. through d. are repeated for each file ASSIGNed in a program. For the structure, meaning, and contents of the FIB and IOAT, reference should be made to the B 2500/B 3500 MCP Information Manual.

The FIB is maintained in the program area, thus, the FIB's for all of a given program's files will be found in the core dump body. Further, all non-disk ASSIGNed files attached to the program at the time of the dump (OPEN or CLOSEd, but not RELEASEd) are reproduced immediately above the core dump body with other pertinent information about the file (file-ID, status, device assignment, etc.).

Following the 200 digit bodies of the FIB are a number of 40 digit entries, one for each I/O area pertaining to that file, which contain the result and the start descriptors for the file buffer as well as other information needed for USE routine exit mechanisms and variable length operations. These areas are shown only in the dump body and the addresses may be used to locate the buffers for the file. In COBOL Programs, file buffers are located beyond the program's code structure. For unblocked, NO ALTERNATE AREA files, the record area and file buffer are one and the same.

The label area for a given file follows the FIB and the "buffer status blocks" and is normally composed of 80 bytes. See the COBOL or MCP manuals for the label format. Disk files, data communications, and unlabeled files have 20-byte label areas.

The file's record (work) area immediately follows the label area.

WORKING-STORAGE ENTRIES.

Immediately beyond the file's work areas is the WORKING-STORAGE SECTION as declared by the COBOL Program.

FIXED SEGMENT CONSTANTS.

Immediately beyond the WORKING-STORAGE entries is the "Compiler's WORKING-STORAGE", or fixed segment constants. This area contains several important areas including:

a. Literals declared in the PROCEDURE DIVISION.
b. ALTER switches; GO TO DEPENDING ON tables.
c. 01 LABEL area.
d. Overlay mechanisms.
e. INDEXes and other uses.

AUXILIARY CODE.

This consists of the following codes, as applicable:

a. Blocking and/or deblocking code for blocked files.

b. Exponentiation code.

c. MONITOR code.

d. USE routines.

OVERLAYABLE CODE.

This consists of user coded areas which were declared to be overlayable.

NON-OVERLAYABLE CODE.

This consists of that part of the Object Program which is always in memory, and cannot be overlaid by any other part of the program.

BUFFERS.

The size and number of input-output buffers depends on the number and type of files.

STACK.

Stack consists of a portion of memory reserved for PERFORM (NTR) subroutine linkage and return mechanisms. Each stack entry created for user coded PERFORMs is 20 digits in length. See appendix I for a more detailed explanation of the use and organization of the program stack mechanism.

DISK FILE HEADERS AND IOAT ENTRIES.

For programs processing disk files, the disk file header and IOAT entry for that file will be found in core memory beyond the Limit register of a given program, and will be shown on the core dump. See the B 2500/B 3500 MCP Information Manual for formats and content explanation of the IOAT. One such entry (each 248 digits long) occurs for each disk file. Note that IOAT entries for disk files are kept beyond the Limit registers. All others are maintained as MCP IOAT table entries within the MCP.

MISCELLANEOUS DUMP LISTING INFORMATION.

A printout of the JRT and MIX entries is provided on a dump listing for a given program. Some of the uses and methods of a dump listing are as follows:

a. Since the dump is written out in digit format and much of the program data is likely to be in byte (DISPLAY) format, a knowledge of the EBCDIC code set, or a table of codes, is needed (see appendix C of this manual).

Little problem should be involved in distinguishing digit and byte information; the addresses shown on the right-side of the COBOL listing can be correlated with the dump. Reference to the COBOL listing will establish the declared format (PICTURE).

b. The heading contains valuable information, including:

1) Program name and mix number.

2) Program status and requirements if not running, e.g., no file.

3) P.A.R. - the next instruction to be executed.

4) Base and Limit registers.

5) Processor toggles.

For some problems, evaluation of a given FIB is valuable, particularly when subscript flags are needed.

Information contained in Program Reserved Memory is frequently very valuable. For example:

a. Switches (where used) are located in locations 0 through 7.

b. Subscript errors may be detected through examination of the Index Registers and the subscript fields.

c. Any GO TO out of range of a PERFORM can be diagnosed through use of Index Register-3, location 40-45, and in conjunction with the stack.

A proper analysis of dumped information requires a compilation program listing to determine the location of files, data areas, and (less frequently) code.

As can be seen, core dumps are a helpful, and at times, a very necessary tool when debugging certain common program errors. Knowledge of the format and content of memory dumps can be fully appreciated only through examination of a number of them. However, core dumps are not the only debugging aid available to the B 2500/B 3500/ B 4500 programmer. A good programmer is aware of all the tools available to him and is able to determine which is best for the evaluation of his problem at hand.

APPENDIX I
B2500/B 3500/B 4500 COBOL
DEBUGGING TECHNIQUES

GENERAL.

This appendix describes the debugging aids available to the programmer for dealing with address errors, instruction errors, and invalid file actions.

ADDRESS ERRORS.

An address error is caused by any of the following conditions:

a. An attempt to address below the Base register or above the Limit register via program stack overflow or subscripting errors.

b. An invalid address in an instruction (i.e., an address containing undigits).

c. An illegal address for a particular instruction (e.g., non-word aligned address for the MOVE words instructions).

Any attempt by a program to execute an instruction containing an address error results in a processor interrupt to the MCP. The MCP analyzes the interrupt, suspends the offending program, and reports the address and disk segment at which the error occurred.

PROGRAM STACK OVERFLOW. This condition is diagnosed by taking a memory dump of the object program at the time of the address error and looking at the addresses of the error (which is usually a PERFORM).

The program stack (the last several hundred digits of a program immediately preceding the limit register) is used by the PERFORM statement, COMPUTE statement (exponentiation), COBOL generated file blocking/deblocking functions, and other routines. Discussion herein will be limited to the actions encountered when using the PERFORM statement.

Every time a PERFORM statement is executed in an object program, an entry will be created in the program stack. Inversely, every time the PERFORM statement is exited from, in the normal way, the entry is removed from the program stack and the next PERFORM statement encountered in the program will create an entry in the same memory location as was previously occupied by the one just removed.

In cases of "nested" PERFORMS, all pertinent entries in the program stack will be created continuously. For example:

	COBOL Statement		Stack Contents
1.		1.	Zeroes (stack empty).
2.	PERFORM A.	2.	Entry for A (stack is 1 deep).
3.	A. EXIT.	3.	Entry A removed (zeroes).
4.	ADD X, Y.	4.	Zeroes (stack empty).
5.	PERFORM B THRU D.	5.	Entry for B (stack is 1 deep).
6.	B. PERFORM C.	6.	Entry for B followed by an entry for C (stack is 2 deep).
7.	C. EXIT	7.	Entry C removed (zeroes) (stack is 1 deep).
8.	ADD X, Y.	8.	Entry for B.
9.	D. EXIT	9.	Entry B removed (zeroes).
10.	GO TO D.	10.	Zeroes (stack empty).

Note that in B 2500/B 3500/B 4500 COBOL the EXIT at D is unnecessary and is shown for documentational purposes only.

Since the program stack is finite, deeply nested PERFORM statements within an object program could fill the stack to an overflow condition, however, such instances can be overcome by increasing the size of the program stack through use of the COBOL MEMORY SIZE clause and recompiling the source program. More commonly, however, the stack is filled due to an incorrect exit from a PERFORMed routine (GO TO out of range of the specified PERFORM) number of times such that the earlier stack entries are not removed. Note that stack entries are removed only by the normal "fall-out" of a PERFORMed routine. Input and output procedures pertaining to SORT statements are considered by the compiler as being PERFORMed routines and will create object code as such. Also, great care should be taken not to go out of the range of the specific PERFORM statement when PERFORMing READ statements which contain AT END or INVALID KEY clauses, or, WRITE statements containing INVALID KEY clauses.

Diagnosis of program stack overflow may be accomplished by:

a. Looking at the program stack in a memory dump (the end of the COBOL listing shows the core location of the beginning of the stack).

b. If the program stack is full (i.e., the locations from the first position of the stack to the end of the program's core area contain non-zero entries - see format below) look for a repetitive pattern.

Program Stack Format

Entry	Size (digits)
Return Control Address (address of the instruction immediately following the PERFORM statement).	6
Index-register-3 (IX3) contents.	8
Hardware indicator setting.	2
Parameter used by the COBOL compiler's EXIT logic.	4

As can readily be seen, a total of 20 digits are used.

c. When a repetitive pattern is found, go to the Return Control Address (as specified in the recurring stack entry) of the compiled COBOL source listing for the given program. The instruction preceding that address will contain a PERFORM. The object code compiled for that PERFORM statement should be examined for out of range GO TO's.

On occasion, the address reported on the SPO by the MCP (using the address error message) is pointing at a COMPUTE statement or, in some cases, at an address preceding the PROCEDURE DIVISION code. The latter case is caused by a compiler generated NTR routine such as required for blocking/deblocking of records, monitor, exponentiation, or overlay code. The fault is still attributable to a GO TO which is out of range of a specific PERFORM statement. While the reported SPO address is worthless in this case, the program stack may still be used to isolate the problem.

SUBSCRIPTING ERROR. A subscript error is caused by increasing the value of the subscript beyond the table size to which it refers, thus, destroying (or using) whatever code is resident in those areas of core. An address error results if program control attempts to execute one of the destroyed core areas and finds an address out of range of the Base and Limit registers. For example:

```
    01 TABLE.
         03 ENTRY PC 9 (6) OCCURS 100 TIMES.
    01 SUBSCRIPT PC 9 (4).
                   .
                   .
                   .
LOOP.  IF ENTRY (SUBSCRIPT) = TEST-VALUE, GO TO FOUND-IT, ELSE ADD
       1 TO SUBSCRIPT THEN GO TO LOOP.
```

Obviously, if an equal condition is not found in the above procedure, the value of SUBSCRIPT will develop beyond the specified 100 times of occurrence and can go as high as 9,999.

If the contents of SUBSCRIPT multiplied by 12 (the length of each TABLE entry in digits), plus the beginning address of TABLE, exceeds the LIMIT register, an address error will result.

If the maximum value of SUBSCRIPT multiplied by the length of the table entry plus the base address of TABLE is not greater than the LIMIT register, then an infinite loop will result and the Overflow Indicator will eventually be turned ON. Note that any attempt to WRITE into a subscripted field when SUBSCRIPT has exceeded the bounds of TABLE can wipe out procedure code, work areas, etc., and usually causes an instruction error. Such a situation can be diagnosed by examining the address of the error in respect to the COBOL source program compilation listing. It will usually consist of an indexing (subscripting) operation.

A memory dump at the time of an address error will provide the value of a given subscript and Index Register-1 (used for the subscripting operation) which is a signed seven (7) digit field (8 digits including sign) starting at base relative location 00008. If the address of the beginning of a table plus IX1 is greater than the end of the program, the cause of the failure is obviously a run-away subscript.

INVALID ADDRESS. An address containing undigits (Hex 10-15) may be accomplished in a COBOL program by attempting to use a subscripting value consisting of those special characters constructed with undigits in the low-order digit position of the byte. An examination of the contents of a given subscript field in a memory dump should immediately show up the problem. Index Register-1 will also contain an undigit(s) in other than the high-order sign position.

The same result will occur when undigits are moved into a given subscript field (e.g., MOVE @FF@ TO SUBSCRIPT, or an equivalent move).

ILLEGAL ADDRESS. This condition should never occur in a COBOL object program. The B 2500/B 3500/B 4500 COBOL Compiler(s) protects against this eventuality by correctly adjusting addresses. Such an occurrence is normally attributed when a COBOL source program is incorrectly using the ENTER SYMBOLIC capabilities of the language.

INSTRUCTION ERROR.

An Instruction Error may be caused by:

a. An attempt to execute an OP CODE which is not legal to the processor, including:

   1) "Impossible" OP codes (those for which no processor instruction exists).

   2) Non-present options (floating point instructions on a machine not equipped with floating point hardware).

   3) Privileged instructions which are designed to be executed only by the MCP.

b. An attempt to execute an instruction for which the required MCP options are not present (e.g., Data Communications instructions when the Data Communications MCP is not in control of the system.)

ILLEGAL OP CODE. An "impossible" OP code may be generated by a "runaway" subscript error, as discussed earlier, where a table is written beyond its limits and thereby wipes out the normal procedure code. The program eventually branches to the altered code and will usually cause the program to halt on an invalid instruction, or an Address Error, or, may even cause an invalid READ(s) or WRITE(s).

Since the contents of the program instruction area of the object code is unpredictable due to the above circumstances, then any of the causes attributed to Instruction Errors can be generated. On very rare occasions, an altered program might not "blow-up" due to its containing no invalid OP codes or addresses in the altered area.

It should be clear that the results of a MOVE into a table, which has been influenced by a "runaway" subscript are unpredictable.

If File Information Blocks are improperly changed, via ENTER SYMBOLIC, then an invalid READ, WRITE, OPEN, or CLOSE most assuredly will occur.

If procedure object code is changed in any way, then an Invalid Instruction or an Address Error can result.

Diagnosis of problems dealing with illegal OP codes requires close examination of a memory dump to find destroyed areas.

INVALID INSTRUCTION. A more common means of generating Instruction Errors is usually seen in programs using the SORT verb in conjunction with an OUTPUT PROCEDURE.

If an Invalid Instruction occurs at an address beyond the address of the last instruction of the program, then the cause is usually due to the passing of program control to the last paragraph of the program and a "fall-out" of that paragraph. The following three examples reflect the most probable cause of invalid instruction occurrences.

Example 1:

```
END-PARA.  CLOSE FILE-A.
END-OF-JOB.
```

Presumably a STOP RUN was neglected in END-PARA.

A relatively rare (but possible case) involves nested PERFORM errors, however, as will be seen in example 3 on the following page, the following situation becomes a more common error in cases where the SORT intrinsic is used.

Example 2:

```
X.  PERFORM A THRU D.
A.  ADD X, Y.
C.  PERFORM B.             B.  ADD M, N THEN GO TO E.
E.  ADD N, Y.
D.  EXIT
END-OF-JOB
```

Note that Paragraph B is incorrect in that a normal COBOL exit from a PERFORMed paragraph is not taken (one must always "fall-out" of a PERFORMed routine).

The stack entry created by the Paragraph C PERFORM statement is not going to be removed, thus, the stack appears as:

Entry of x	Entry of C

and the stack pointer (Index Register-3), which normally points to the currently-in-use stack entry, is in fact, pointing at the "Entry of C".

The subroutine exit mechanism created in object programs by the compiler involves checking the currently-in-use stack entry to see if that entry "is mine" - i.e., does the PERFORM statement which created the current entry correspond to the PERFORM statement of this particular routine? This permits the complex overlapping of PERFORM statements which are acceptable to the B 2500/B 3500/B 4500.

Since the current stack entry in the preceding example (Entry of C) does not correspond to the PERFORM statement of the X routine, the exit back to Paragraph A which follows paragraph X is not taken and "fall-thru" occurs. An Instruction Error (Invalid Instruction) is likely to occur at this point due to the fact that the next "instruction" is in fact the program's I/O buffers or stack area.

A GO TO which is out of range of a nested PERFORM is much more likely to occur in the program body than at the extreme end of the program and in that case, unpredictable results will be detected in the program's execution.

Example 3:

When using the SORT verb, it seemingly is common practice to place the OUTPUT PROCEDURE pertaining to a sort as the last routine in a given program. Since the OUTPUT PROCEDURE is PERFORMed by the SORT statement (i.e., it is executed as a subroutine and the normal PERFORM exit mechanism is generated at the end of the OUTPUT PROCEDURE SECTION), therefore, any GO TO out of the range of a PERFORM within the OUTPUT PROCEDURE will also cause failure of a proper exit at the end of the OUTPUT PROCEDURE. Results will be as described for Case (B). This discussion also holds when the INPUT PROCEDURE is the last routine in a given program containing a SORT statement.

Diagnosis for the preceding example involves:

a. Recognizing that the invalid instruction address resides beyond the final executable instruction of the program, and:

1) Examination of the current stack entry in the memory dump. This entry can be found by going to the address reflected in Index Register-3 (core location 24) which is the current stack being referenced.

2) The first six digits of the current stack location will pin-point the return control address (i.e., the address of the instruction which immediately follows the offending PERFORM). Reference to the COBOL listing, using the return control address, will identify the PERFORM statement causing the trouble. The object of that PERFORM should be examined for GO TO errors.

Note that the stack should contain at least two entries if this condition exists. The first is a result of PERFORMing the OUTPUT PROCEDURE by the SORT and the last entry is the address of the offending PERFORM. If more than two stack entries exist, then more than one GO TO "out-of-range" is involved.

There may be other stack entries beyond the "current" entry to which Index-Register-3 is pointing, however, these are not "live" (current) entries and should be ignored in as much as they have already been "removed" by earlier exit mechanisms.

b. An invalid instruction caused by the lack of floating point hardware can occur in a COBOL program compiled with the COBOL compiler only if the program uses exponentiation (e.g., COMPUTE A = B**C) and the floating point hardware is not an installed feature of the processor.

c. An invalid instruction due to an attempt to execute a privileged instruction should never occur in a COBOL program.

NOTE

All of the above could occur if the program's code area has been destroyed by subscript errors.

INVALID FILE ACTIONS (OPEN, CLOSE, READ, WRITE).

In all cases of "DS" conditions, the MCP displays on the SPO the pertinent program name, its mix-index, program address register (P.A.R., e.g., the address of the next instruction to be executed), the segment of the program being executed and, in the case of invalid file actions, the file-name about which the file action is concerned.

For COBOL programs, the segment number reported is three (3) greater than the actual segment number. Further, the segment number is not the number following the SECTION statement, e.g., ABLE SECTION 50; it is the segment number found on the right side of the COBOL listing just prior to the program address for that code line. The segment number assigned to the resident portion of a program is not shown on the listing. The MCP reports it as being segment number three (003).

For invalid file actions, the P.A.R. reflects the address of an unconditional branch instruction (BUN) following the branch communicate (BCT) for that operation. All BCT's (MCP calls) are of the form:

```
BCT nnnn (nnnn = address in MCP)
BUN X
[Parametric information]
x.(instruction following parameters)
```

The first parameter (6 digits) for all file actions points to the file information block (FIB) pertaining to that file. The other parameters vary with the type of action to be performed and are discussed below.

The address of the FIB can be found on the right of the COBOL listing on the line pertaining to the FD for that file.

The FIB can be very helpful in diagnosing the precise circumstances of a given problem. Frequently the problem is obvious, but since each illegal file action has more than one possible cause, the exact problem may be elusive. The most helpful portion of the FIB is FIBSTA, the sixteenth (16) digit of the FIB which contains file status information in the following combinations:

0	= Open.
1	= Closed; never opened.
2	= Restricted - generally, END of file sensed.
3	= Closed; previously opened.
5 or 7	= Opening next reel (tape).
9	= Multi-file search in progress.

Generally, only the first four codes concern the programmer while debugging. Examples follow:

a. Invalid file OPEN - A request to OPEN a file generates:

```
BCT
BUN
```

xxxxxx	FIB address for file
y	OPEN type (I, O, I/O or O/I)
z	OPEN variant (NO REWIND, REVERSE, etc.)

An invalid file OPEN is most commonly due to an attempt to OPEN a file that had been previously OPENED and not yet CLOSED.

b. Invalid file CLOSE - A request to CLOSE a file generates:

BCT	
BUN	
xxxxxx	FIB address for file
y	Variant (RELEASE, LOCK, PURGE, etc., REEL or FILE)
z	DCOM only

An invalid CLOSE is generally due to a program call to CLOSE a file which has never been OPENed or that is already CLOSEd. As with an invalid OPEN, the program output can frequently be used to distinguish the possible causes; if any confusion exists FIBSTA should be examined for a 1 (never OPENed) or 3 (previously OPENed).

c. Invalid file READ - A READ generates:

BCT	
BUN	
xxxxxx	FIB address
YYYYYY	End of file branch address

An invalid file READ is generally due to an attempt to READ a file which is not OPEN or to READ beyond End-of-File. Less frequently, it is due to an attempt to READ a file which cannot be read (e.g., a print file or a tape which had been OPENed as OUTPUT).

These may be distinguished by FIBSTA: 1 or 3 indicates the file to be CLOSED; 2 indicates restricted (EOF sensed) and 0 indicates the file to be OPEN.

NOTE

For variable length records, an additional condition exists which is explained in example d that follows.

d. Invalid file WRITE - A WRITE generates:

BCT	
BUN	
xxxxxx	FIB address
YYYYYY	for disk, invalid key branch; for branch (assembler only), end-of-page branch
uuvv	For printer, space and skip variants

An invalid file WRITE is generally due to an attempt to WRITE a file which is not OPEN; or, when working with a sequential disk file, an attempt to WRITE a file after the INVALID KEY branch has been taken; or less frequently, an attempt to WRITE a file which cannot be written on (e.g., a card reader file, or, a tape file OPENed as INPUT).

FIBSTA can be used to distinguish between an attempt to WRITE a file before it is OPENed or after it is CLOSEd; further, a WRITE initiated after End-of-File can be determined by looking at the restricted flag in FIBSTA.

Another method of generating an invalid file WRITE pertains to variable-length records (FIBBLK, the 45th digit of the FIB can be referenced for a digit 2 code, which indicates variable-length records).

The first four bytes of a variable-length record <u>must</u>

contain a modulo-2 (even) count of the number of bytes in the record including the four byte count indicator (the record must be composed of an integral number of words).

Further, the count may not exceed the declared maximum record size. If either condition is not met, an invalid WRITE occurs. The problem may be diagnosed by noting:

1) FIBSTA contains a zero (file OPEN).

2) FIBBLK contains a 2 (variable-length).

3) The first four bytes of the record are invalid (the number is odd, zero or exceeds FIBMRL divided by 2 (FIBMRL, positions 20-24 in the FIB is the maximum record length, in digits)).

In all cases, FIBSTA can help to narrow down the cause of the problem. The exact logic error which led to the invalid operation can be as simple as a failure to OPEN the file. However, a knowledge of the file status at the time the MCP detected the error is very often the key to what has happened.

e. Invalid File Required - This is rarely seen and is due to an attempt to OPEN a device in the wrong mode (e.g., OUTPUT on the card reader, INPUT on the printer).

f. EOF-No Label - This message is caused by READing a random disk file address into an area not yet assigned. Assume a random file with paging of 5 by 1000 records. Initial file creation OPENed pages 1,2,3, and 5. If a record with a key of 4000 to 4999 is READ, the INVALID KEY branch of the READ statement becomes effective. However, if an INVALID KEY branch was not coded by the programmer, an EOF NO LABEL occurs. This message is accompanied by the MCP's DS or DP message. A READ, on any sequential file, without an AT END option can also cause this error.

INDEX

Revised 9/28/70
by PCN 1033099-004

INDEX (cont'd)

INDEX (cont'd)

INDEX (cont'd)

INDEX (cont'd)

BURROUGHS CORPORATION
DATA PROCESSING PUBLICATIONS
REMARKS FORM

TITLE: ______________________________
____
____

FORM: ____________________
DATE: ____________________

CHECK TYPE OF SUGGESTION:

□ ADDITION □ DELETION □ REVISION □ ERROR

GENERAL COMMENTS AND/OR SUGGESTIONS FOR IMPROVEMENT OF PUBLICATION:

FROM: NAME ______________________________
TITLE ______________________________
COMPANY ______________________________
ADDRESS ______________________________
____

DATE ____________________

tear along dotted line

STAPLE

FOLD DOWN SECOND FOLD DOWN

Postage
Will Be Paid
by
Addressee

No
Postage Stamp
Necessary
If Mailed in the
United States

BUSINESS REPLY MAIL
First Class Permit No. 817, Detroit, Mich. 48232

Burroughs Corporation
6071 Second Avenue
Detroit, Michigan 48232

attn: Sales Technical Services
Systems Documentation

FOLD UP FIRST FOLD UP

Wherever There's
Business There's Burroughs

1033099 5-69 Printed in U.S. America