Burroughs

B 7000/B 6000 Series

COBOL

REFERENCE MANUAL

PRICED ITEM

Printed in U.S.A.

August 1977

5001464

Burroughs

B 7000/B 6000 Series

COBOL

REFERENCE MANUAL

PRICED ITEM

Printed in U.S.A.

August 1977

5001464

Burroughs believes that the software described in this manual is accurate and reliable, and much care has been taken in its preparation. However, no responsibility, financial or otherwise, can be accepted for any consequences arising out of the use of this material, including loss of profit, indirect, special, or consequential damages. There are no warranties which extend beyond the program specification.

The Customer should exercise care to assure that use of the software will be in full compliance with laws, rules, and regulations of the jurisdictions with respect to which it is used.

The information contained herein is subject to change. Revisions may be issued from time to time to advise of changes and/or additions.

Correspondence regarding this document should be addressed directly to Burroughs Corporation, P. O. Box 4040, El Monte, California 91734, Attn: Publications Department, TIO – West.

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)

TABLE OF CONTENTS (Cont)

TABLE OF CONTENTS (Cont)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (Cont)

LIST OF TABLES

ACKNOWLEDGEMENT

The information contained in this document is based on the COBOL language initially developed in 1959 and subsequently updated.

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

The authors and copyright holders of the copyrighted material used in this document,

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

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

Any organization interested in reproducing the COBOL report and specifications in whole or 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 a 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.

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

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

1. INTRODUCTION

This manual provides a complete description of COBOL (Common Business Oriented Language) as implemented on the Burroughs B 7000/B 6000. The specifications contained in this manual describe all options of the language which are accepted by the B 7000/B 6000 COBOL compiler.

COBOL is a machine-independent, procedural language for describing computer programs. The language is designed so that a COBOL program reads much like ordinary English and, therefore, automatically provides much of its own documentation. For a description of the official language specifications, see the Journal of Development, published by the CODASYL COBOL Programming Language Committee. For a description of American National Standard COBOL, see X3.23-1968 and 1974 published by the American National Standards Institute.

B 7000/B 6000 COBOL is an extended version of this language which provides the user with the complete facilities of the B 7000/B 6000 hardware-software system without resorting to machine or assembly languages. The following areas of B 7000/B 6000 COBOL are extensions to the above COBOL specifications:

Relaxation of margin restrictions.

The CONFIGURATION SECTION is optional.

The DISK SIZE clause of the OBJECT-COMPUTER paragraph.

SPECIAL-NAMES entries are, in some cases, unique to B 7000/B 6000 COBOL.

The END option of the FILE-LIMITS clause.

Dynamic allocation of buffers with the data-name option of the RESERVE clause.

The PREPARED FOR clause of the DATA DIVISION header.

The WORDS option of the BLOCK CONTAINS and RECORD CONTAINS clauses.

The LABEL RECORDS clause which specifies the WITH MULTIPLE AT END phrase.

The specification of character set in the RECORD CONTAINS clause.

The SAVE-FACTOR and VALUE OF ID clauses.

The abbreviations ID, VA and OC.

The LOWER-BOUNDS, RANGE, RECORD AREA and SEGMENT clauses.

The OCCURS clause at the 01 level.

The definition of tables of more than three dimensions.

The PICTURE character "J".

The use of non-PICTURE characters in a PICTURE entry as fixed insertion characters.

The USAGE options; ASCII, COMP-1, COMP-2, COMP-4, COMP-5 and DISPLAY-1.

The LINKAGE SECTION.

The special registers TIME, COMPILETIME, CHECKPOINT-STATUS and TODAYS-DATE.

The use of arithmetic expressions and subscripted variables as subscripts.

The arithmetic operators MOD and DIV.

The arithmetic functions.

The logical association of ELSE with the AT END, EOP, INVALID KEY and ON SIZE ERROR clauses.

Arithmetic operands may be a maximum of 23 digits instead of only 18.

The word TO is optional in the EQUAL TO relational operator and in the GO TO statement.

The DUMP, MERGE and MONITOR statements.

The CRUNCH, HERE, PURGE and RELEASE options of the CLOSE statement.

The use of a formula in the DEPENDING option of the GO TO statement.

The HERE option of the EXIT statement.

The relational operator UNEQUAL.

The type c abbreviation of conditional statements.

Partial word moves (options three and four of the MOVE statement).

The REEL-NUMBER option of the OPEN statement.

In the PERFORM statement:

- The use of a formula in the VARYING option.
- No limit on the number of AFTER phrases.
- No restrictions on the range of a nested PERFORM.

The SEEK statement for sequential disk files.

The AFTER RECORD SIZE ERROR option of the USE statement.

The CHANNEL, AUXILIARY, ALTERNATE and ERROR options of the WRITE statement.

The facility for handling Data Communications files in the same manner as any other type of file.

The use of file attributes for dynamic definition of file characteristics.

The facility for handling any type of file as a DIRECT file thus bypassing most of the MCP procedures for file and record processing.

The facility for invoking user written intrinsics.

All syntax for handling interrupt procedures.

All syntax for handling Inter Program Communication, synchronous processes and asynchronous processes.

All syntax for handling switch files.

All syntax for interface with the Burroughs Data Management System.

Selective implementations of statements, clauses, and phrases contained within the ANSI 74 Indexed I-O, Sequential I-O, Table Handling, and Nucleus Modules. (Refer to Appendices B, D and E).

Various B 3700 system compatibility features implemented under the auspices of the "B2500" system dollar option (Refer to Appendices C and D).

B 7000/B 6000 COBOL utilizes the B 7000/B 6000 features of automatic program segmentation, automatic peripheral assignment, virtual memory/storage allocation, multiprocessing, re-entrant programming and debugging language statements to allow a high degree of sophistication in application program design and development.

ADVANTAGES OF COBOL

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 programming 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 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 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.

PROGRAM ORGANIZATION

Every COBOL program must contain these four divisions in the following order:

```
IDENTIFICATION
ENVIRONMENT
DATA
PROCEDURE
```

The IDENTIFICATION DIVISION identifies the program. In addition, the programmer may include such optional pieces of information as the date compiled, and programmer's name for documentation purposes. This division is completely machine-independent and thus does not produce object code.

The ENVIRONMENT DIVISION specifies the equipment being used. It contains computer descriptions and deals, to some extent, with the files the program will use.

The DATA DIVISION contains file and record descriptions describing the data files that the object program is to manipulate or create, and the individual logical records which comprise these files. The characteristics or properties of the data are described in relation to a standard data format rather than an equipment-oriented format. Therefore, this division is to a large extent computer-independent. While compatibility among computers cannot be absolutely assured, careful planning in the data layout will permit the same data descriptions, with minor modification, to apply to more than one computer.

The PROCEDURE DIVISION specifies the steps that the user wishes the computer to follow. These steps are expressed in terms of meaningful English words, statements, sentences, and paragraphs. This division of a COBOL program is often referred to as the "program" itself. In reality, it is only part of the total program, and is insufficient by itself to describe the entire program. This is true because repeated references must be made (either explicitly or implicitly) to information appearing in the other divisions. This division, more than any other, allows the user to express his/her thoughts in meaningful English. Concepts of verbs to denote actions, and sentences to describe

procedures, are basic, as is the use of conditional statements to provide alternative paths of action.

A program written in COBOL is called the source program, and is accepted as input by the B 7000/B 6000 COBOL compiler. The compiler will verify that the rules presented in this manual have been followed and will generate an object program in machine code, ready to be executed. Due to the speed of compilation, no object deck is supplied. Instead, the object program is placed on the disk, and may be dumped on a magnetic tape for back-up storage. Should changes become necessary, the source deck is corrected and a new compilation run made. Thus, the source deck always reflects the object program being executed.

OTHER PUBLICATIONS

Additional information which may be of interest to the COBOL programmer may be found in the following publications:

1058633	B 6700 Information Processing System Reference Manual
5000060	Data Communications Functional Description
5001563	B 7000/B 6000 System Software Operational Guide, Volume 1, including the following: Indexed Sequential Access Method (ISAM) Mathematical Intrinsics SORT
5001779	B 7000/B 6000 Input/Output Subsystem Reference Manual
5000722	System Software Handbook
5001456	B 7000/B 6000 System/Binder Reference Manual
5001092	DMSII, Host Language Reference Manual
5001175	Workflow Management Reference Manual
5001290	B 6800 System Reference Manual

2. LANGUAGE FORMATION

GENERAL

This section discusses the elements of which the B 7000/B 6000 COBOL language is constructed and the rules governing this construction. Being composed of words, statements, sentences, paragraphs, and so forth, COBOL is seen to be a language based structurally upon English.

CHARACTER SET

The basic elements of any language are the characters of which it is formed. The B 7000/B 6000 COBOL character set consists of the following 54 characters:

0-9		;	Semicolon
A-Z		.	Period (decimal point)
Space or blank		"	Quotation mark
-	Minus sign	(	Left parenthesis
+	Plus sign	)	Right parenthesis
*	Asterisk	>	Greater than symbol
/	Slash (virgule)	<	Less than symbol
=	Equal sign	[	Left bracket
$	Currency sign	]	Right bracket
,	Comma	:	Colon

The left bracket, right bracket, and colon are Burroughs extensions to the standard COBOL character set.

Alphabetic Characters

An alphabetic character consists of any of the following:

A - Z
Space or blank

Numeric Characters

A numeric character consists of any of the digits 0 thru 9.

Alphanumeric Characters

An alphanumeric character is any character belonging to the B 7000/B 6000 character set.

Editing Characters

An editing character is any single or fixed two-character combination of the following set:

B	Space or blank insert
Z	Zero suppress
0	Zero insert
+	Plus
-	Minus
CR	Credit
DB	Debit
*	Check protect
$	Currency sign
,	Comma
.	Period

Punctuation Characters

A punctuation character is any character of the following set:

,	Comma
;	Semicolon
:	Colon
.	Period
"	Quotation mark
(	Left parenthesis
)	Right parenthesis
	Space or blank
[	Left bracket
]	Right bracket
/	Slash (virgule)

Punctuation characters and one or more spaces may be used as separators and with character strings as follows:

a A character-string is delimited on the right by a space or by any special character except the hyphen.

b. One or more spaces may immediately precede and/or follow any delimiter.

c. A punctuation character may be preceded and/or followed by one or more spaces except when restricted by special insertion editing in the PICTURE clause.

Relation Characters

The B 7000/B 6000 COBOL compiler accepts the following characters in conditional relations:

>	Greater than symbol
<	Less than symbol
=	Equal sign

Characters Used for Words

The character set for words consists of the following characters:

0 - 9	
A - Z	
-	Hyphen

LANGUAGE DESCRIPTION NOTATION

COBOL reference manuals have almost universally adopted a particular form of notation. This manual uses that notation as described in the paragraphs that follow.

Key Words

All underlined upper case words are key words and are required when the functions of which they are a part are used. An error will occur at compilation time if the underlined words are absent or misspelled. Examples of key words would be:

```
SAVE-FACTOR

FILE-LIMIT
```

Optional Words

All words not underlined are optional and are included for readability only. They may be included or excluded in the source program. If they are included, they must be spelled correctly. For example, use of the reserved words MODE and IS are optional in the RECORDING MODE clause. The clause might be written as:

```
        RECORDING MODE IS STANDARD

or as:

        RECORDING MODE STANDARD

or as:

        RECORDING STANDARD
```

Generic Terms

Lower case words written in a format are generic terms, indicating the type of word which must be supplied in that format position by the programmer. Examples of this would be:

```
RECORD CONTAINS  integer  CHARACTERS
66  data-name-1  RENAMES  data-name-2
```

Brackets and Braces

When a portion of a general format is enclosed in brackets, that portion may be included or omitted at the user's choice.

Braces, { }, enclosing a portion of a general format denote that a selection of one of the options contained within the braces must be made. In both cases, a choice is sometimes indicated by vertically stacking the possibilities. When brackets or braces enclose a portion of a format, but only one possibility is shown, the function of the brackets or braces is to delimit that portion of the format to which a following ellipsis applies.

Ellipsis

The appearance of three consecutive periods (...) within any format indicates that the data bounded by the right bracket, brace, or parenthesis delimiter immediately preceding the periods and its logically matching left bracket, brace, or parenthesis delimiter may be successively repeated, depending on the requirement of the user.

DEFINITION OF WORDS

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

A	- Z
0	- 9
-	Hyphen

A word may not begin or end with a hyphen, and a space is not an allowable character in a word. A word is ended by a space or any of the special characters except the hyphen.

TYPES OF WORDS

COBOL contains three basic word types (i.e., nouns, verbs, and reserved words) which are described as follows:

Nouns

Nouns are divided into the following 12 special categories; a brief description is provided for each category:

- Data-Name
- File-Name
- Record-Name
- Condition-Name
- Procedure-Name
- Mnemonic-Name
- Index-Name
- Literal
- Figurative Constant
- Special Registers
- Event-Name
- Lock-Name

Data-Name. A data-name is a single word and must contain at least one alphabetic character. The data-name is used to refer to an item of data, or to a defined area containing the data.

Example:

```
STOCK-NAME-2
```

File-Name. A file-name is a single word and must contain at least one alphabetic character. File-names are used to reference a file.

Record-Name. A record-name is a user defined word containing at least one alphabetic character, used to identify a logical record. A record may be subdivided into several data items, each of which is identified by a data-name.

Condition-Name. A condition-name is the name assigned to a specific value, set of values, or range of values, within the complete set of values that a data item may assume. The data item itself is called a "conditional variable." The condition-name must contain at least one alphabetic character and must be unique

or be able to be referenced uniquely thru qualification. A conditional variable may be used as a qualifier for any of its condition-names. If references to conditional variable require indexing, or subscripting, then references to any of its condition-names also require the same combination of indexing, or subscripting. A condition-name is used in conditions as an abbreviation for the relation condition; its value is TRUE if the associated condition variable is equal to one of the set values to which that condition-name is assigned.

Procedure-Name. A procedure-name is either a paragraph-name or section-name and may consist entirely of numeric characters. Procedures are referred to by use of the procedure-name. Numeric procedure-names are equivalent if, and only if, they are composed of the same number of digits and have the same value.

Literal. A literal is an item of data whose value is identical to the characters contained within the item. There are three classes of literals: numeric, floating-point and non-numeric.

a. Numeric Literal. A numeric literal is defined as an item composed of characters chosen from the digits 0 thru 9, the plus sign or minus sign, and the decimal point.
An integer numeric literal is a string of digits which may optionally contain a + or - as the leftmost character and no digits to the right of the decimal point location. A non-integer numeric literal must contain a single decimal point.

Examples:	Integer Numeric Literal	Non-Integer Numeric Literal
	+601	+98.6
	-234	234.8
	0	.005
	125	-.1

The number of digits in a numeric literal may not exceed 23.

If a literal conforms to the rules for the formation of numeric literals, but is enclosed in quotation marks, it is a non-numeric literal and is treated as such by the compiler.

b. Floating-Point Literal. A floating-point literal is intended for use with COMP-4 and COMP-5 data items as an alternative to using a standard numeric literal. The format of a floating-point literal is:

$$\left[\begin{matrix}+\\-\end{matrix}\right] \text{ mantissa } \underline{\text{E}} \left[\begin{matrix}+\\-\end{matrix}\right] \text{ exponent}$$

The mantissa may be signed and must have one decimal point. The exponent may be signed and must be an integer.

For single precision, the range of permissable values is $8.7581154020 \times 10^{-47}$ to $4.31359146673 \times 10^{68}$. For double precision, the range of permissable values is $1.9385458571375858335564 0 \times 10^{-29581}$ to $1.9488283820502807912 4469 \times 10^{29603}$. Zero is also permissable for either single or double precision.

Floating-point literals may be used any place in the langauge where a non-integer numeric literal is permitted, except that floating-point literals may not be moved to non floating-point data items nor may they be used in a relation condition involving a non-numeric data item.

Examples:

```
1.E-40
-.0023E29
+.0012345E-5
+1.2E9500
2.E40
+123.45678901234E20
```

If a literal conforms to the rules for the formation of floating-point literals, but is enclosed in quotation marks, it is a non-numeric literal and is treated as such by the compiler.

A floating-point literal may not have embedded blanks. If the mantissa and/or the exponent are unsigned, they will be assumed to be positive.

c. Non-Numeric Literal. A non-numeric literal may be composed of any character of the B 7000/B 6000 standard character set. The non-numeric literal must be enclosed within quotation marks, and any spaces contained within the quotation marks are considered to be a part of the non-numeric literal and are therefore part of the value. Within the literal, each set of two contiguous quote marks represents a single quote mark. The non-numeric literal itself cannot exceed 256 characters in length.

Examples:

```
"2E50"
"+12.3"
"BC.D"
"ZERO"
"B. W. ""BILL"" JONES"
```

d. Undigit Literal. Hexadecimal literals (UNDIGIT literals) can be used in comparisons and MOVEs involving unsigned integer four bit character numeric data items (or as the initial value of those data items). Undigit Literals must be bounded on both ends by the character "@" and must contain only the hexadecimal characters 0 through F. In addition, an Undigit Literal may be preceded by the figurative constant ALL. An Undigit Literal may not be used in an arithmetic statement, and may not be moved to an item which requires scaling or editing. This feature is available only while the B2500 system dollar option is set. Refer to Appendix C for a description of the B2500 implementations.

Examples:

@00CDAEF@

@C@

Figurative-Constant. A figurative constant is a particular value that has been assigned a fixed name and may be used in any place in which a literal would be allowed. When a figurative constant is used to represent the corresponding value, it must never be enclosed in quotation marks. This does not preclude the use of a figurative constant as a non-numeric literal when it must be enclosed in quotation marks. The figurative constant names and their meanings are shown in the following table.

FIGURATIVE CONSTANTS	MEANING
ZERO ZEROS ZEROES	Represents the value 0, or one or more of the character 0, depending on the context.
SPACE SPACES	Represents one or more spaces or blanks.
HIGH-VALUE HIGH-VALUES	Represents the highest value in the collating sequence. (See figure 2-1).
LOW-VALUE LOW-VALUES	Represents the lowest value in the collating sequence. (See figure 2-1).
QUOTE QUOTES	Represents one or more of the single character ". The word QUOTE may not be used to bound a non-numeric literal.
UPPER-BOUND UPPER-BOUNDS	Represents the highest value allowable within the usage of a data item.

FIGURATIVE CONSTANTS	MEANING
LOWER-BOUND LOWER-BOUNDS	Represents the lowest value allowable within the usage of a data item.
ALL literal	Represents one or more of the string of characters comprising the literal. The literal must be either a non-numeric literal or a figurative constant other than ALL. When a figurative constant is used, the word ALL is redundant and is used for readability only.

When a figurative constant represents a string of one or more characters, the length of the string is determined by the compiler from context according to the following rules:

a. When a figurative constant is associated with another data item, as when the figurative constant is moved to or compared with another data item, the string of characters specified by the figurative constant is repeated character by character until the size of the resultant string is equal to the size in characters of the associated data item. ALL must be followed by a non-numeric literal or a figurative constant. ALL followed by a numeric literal is illegal.

Example 1:

```
MOVE ALL "A" TO YEAR.
```

Where YEAR has been described as having four characters, AAAA would result.

Example 2:

```
MOVE ALL "NO-OP" TO FLIGHT-PATTERN.
```

Where FLIGHT-PATTERN has been described as having 12 characters, NO-OPNO-OPNO would result.

Example 3:

```
MOVE SPACES TO HEADING.
```

The area defined as HEADING would contain as many spaces as indicated by the SIZE or PICTURE clause used to describe HEADING in the DATA DIVISION.

b. When a figurative constant is not associated with another data item, as when the figurative constant appears in a DISPLAY or EXAMINE statement, the length of the string is one character. The figurative constant ALL literal may not be used with DISPLAY or EXAMINE.

A figurative constant may be used any place where a literal appears in the format, except that the figurative constant, like its literal equivalent, must not contradict the usage specified for the data item where it is used. (See figure 2-1.)

Edited numeric items are treated as alphanumeric for comparisons.

Moving ZEROS, UPPER-BOUNDS, or LOWER-BOUNDS to a group of COMP-2 usage resets the digit in any field subordinate to the group.

Example 4:

MOVE ALL QUOTES TO YEAR.

Where YEAR is four characters, """" would result.

Figure 2-1 defines the characters used for each of the figurative constants. The notation of 4"FF" means that the 4-bit (HEX) characters FF are used; 6" " indicates that the BCL representation of a space is used; 7" " indicates that the ASCII representation of a space is used; 8" " indicates that the 8-bit (EBCDIC) representation of a space is used. The abbreviations for the type column are AN for alphanumeric, EA for edited alphanumeric, AB for alphabetic, NM for numeric and EN for edited numeric. Note that this chart indicates the characters or values used when a data item of a given usage and category is used in a relation condition with a given figurative constant.

	TYPE	ZEROS	LOWER BOUNDS	UPPER BOUNDS	HIGH VALUES	LOW VALUES	SPACES	QUOTES
COMP-2	GROUP ELEM.	0 CHR. 0 VALUE	0 CHR. 0 VALUE	9 CHR. 9's VALUE	* *	* *	* *	* *
COMP or COMP-1	GROUP ELEM.	0 0	0 0	* **	* *	* *	* *	* *
EBCDIC	AN EA AB NM EN	8"0" 8"0" 8"0" 0 VALUE 0 VALUE	4"00" 4"00" 8" " 0 VALUE 0 VALUE	4"FF" 4"FF" 8"Z" 8"9" 8"9"	4"FF" 4"FF" 4"FF" 4"FF" (1) 4"FF" (1)	4"00" 4"00" 4"00" 4"00" (1) 4"00" (1)	8" " 8" " 8" " 8" " (2) 8" " (2)	8"""" 8"""" 8"""" 8"""" (1) 8"""" (1)
BCL	AN EA AB NM EN	6"0" 6"0" 6"0" 0 VALUE 0 VALUE	6" " 6" " 6" " 0 VALUE 0 VALUE	6"9" 6"9" 6"Z" 6"9" 6"9"	* * * * *	* * * * *	6" " 6" " 6" " 6" " (2) 6" " (2)	6"""" 6"""" 6"""" 6"""" (1) 6"""" (1)
ASCII	AN EA AB	7"0" 7"0" 7"0"	4"00" 4"00" 7" "	4"FF" 4"FF" 7"Z"	4"FF" 4"FF" 4"FF"	4"00" 4"00" 4"00"	7" " 7" " 7" "	7"""" 7"""" 7""""

* = INVALID ** Maximum Integer Value, Single or Double.
See notes (1) and (2) on following page.

Figure 2-1. Use of Figurative Constants

The rules for moving a figurative constant are somewhat different when a figurative constant having an implied alphabetic or alphanumeric category is moved to a numeric or numeric edited data item. (See notes ① and ②.)

Note①: The figurative constants high-value, low-value, and quote have an implied alphanumeric category. Therefore, when they are compared in a relation condition with a numeric data item, the comparison is done according to the rules for a non-numeric comparison. When these figurative constants are moved to a numeric or numeric-edited data item, the move is done according to the rules for moving an alphanumeric item to a numeric or numeric-edited item. i.e., the results are the same as moving an alphanumeric data item containing the figurative constant in all character positions. Because the data is moved as if the sending item was described as an unsigned numeric integer, non-numeric characters in the source are converted to numeric characters. This results in high-value EBCDIC characters being converted to EBCDIC "9", low-value EBCDIC characters being converted to EBCDIC "0", EBCDIC quote characters being converted to EBCDIC "7", and BCL quote characters being converted to BCL "9".

Note②: The figurative constant spaces has an implied category of alphabetic. It is illegal to move spaces to a numeric or numeric-edited data item.

Special Registers. There are several special registers available in B 7000/B 6000 COBOL. These are as follows:

- TALLY
- LINE-COUNTER
- PAGE-COUNTER
- CHECKPOINT-STATUS
- LINAGE-COUNTER
- TODAYS-DATE
- TIME (n)
- COMPILETIME (n)

a. TALLY. The function of TALLY is to hold information produced by the EXAMINE statement; however, it may be used by a program as temporary storage whenever the EXAMINE statement is not being used. The implicit description of TALLY is PIC 9(11), and it is maintained in COMPUTATIONAL-1 form.

b. LINE-COUNTER. The word LINE-COUNTER is the fixed data-name for a COMPUTATIONAL LINE-COUNTER that is generated for each Report Description in the Report Section to determine the vertical positioning of a report. One LINE-COUNTER is automatically supplied for each report described in the REPORT SECTION if a PAGE LIMIT clause is included in the report description entry.

c. PAGE-COUNTER. The word PAGE-COUNTER is a fixed data-name for a COMPUTATIONAL PAGE-COUNTER that is generated for each Report Description entry in the Report Section for use as a source data item for page numbers within a report group. One PAGE-COUNTER is supplied for each report for which the word PAGE-COUNTER is included as a source data item in a report group description entry.

d. CHECKPOINT-STATUS. The word CHECKPOINT-STATUS is the fixed data-name used by the CHECKPOINT/RERUN facility. See the discussion of CHECKPOINT in Section 7 for a discussion of CHECKPOINT-STATUS.

e. LINAGE-COUNTER. The word LINAGE-COUNTER is a fixed data-name for a COMPUTATIONAL line counter generated by the presence of a LINAGE clause in a File Description. The implicit class of a LINAGE-COUNTER is numeric. The value represented in the LINAGE-COUNTER at any given time is the number of lines advanced within a printed page. One LINAGE-COUNTER is supplied for each file in the FILE SECTION whose FD entry contains a LINAGE clause.

f. TODAYS-DATE. TODAYS-DATE is synonymous with TIME(15) and will return the current date in DISPLAY form in the format "MMDDYY".

g. TIME(n). Various times can be made available to a COBOL program by the use of TIME(n), where n must be an integer ranging from 0 thru 15:

- TIME(0). Returns the current Julian date, in the form "YYDDD", where YY is the last two digits of the year and DDD is the day of the year, in DISPLAY-1 form.
- TIME(1). Returns in COMPUTATIONAL form as an integer value the time of day in sixtieths of a second.
- TIME(2). Returns in COMPUTATIONAL form as an integer value the elapsed processor time of the program in sixtieths of a second.
- TIME(3). Returns in COMPUTATIONAL form as an integer value the elapsed I/O time of the program in sixtieths of a second.
- TIME(4). Returns in COMPUTATIONAL form as an integer value the contents of a 6-bit machine clock which increments every sixtieth of a second.

- TIME(5). Returns the current date in the format "MMDDYY", where MM is the month, DD is the day, and YY is the last two digits of the year, in DISPLAY-1 form.
- TIME(10). This is the same as TIME(0) except the value is returned in DISPLAY form rather than DISPLAY-1.
- TIME(11). This is the same as TIME(1) except the time is in increments of 2.4 microseconds rather than sixtieths of a second.
- TIME(12). This is the same as TIME(2) except the time is in increments of 2.4 microseconds rather than sixtieths of a second.
- TIME(13). This is the same as TIME(3) except the time is in increments of 2.4 microseconds.
- TIME(14). Returns in COMPUTATIONAL form as an integer value, the contents of a 36-bit machine clock which increments every 2.4 microseconds. The contents of the machine clock do not necessarily contain the current time of day and should be used for interval timing purposes only.
- TIME(15). Returns the current date in the format "MMDDYY", where MM is the month, DD is the day, and YY is the last two digits of the year, in DISPLAY form.
- COMPILETIME(n). COMPILETIME(n) is similar to TIME(n). The parameter range n is the same, but COMPILETIME(n) is not dynamic and it returns the values of TIME(n) as they existed at compile time, thus enabling the object program to find out when it was compiled and how long it took.

NOTE: Most special registers have an implicit class of numeric. Numeric special registers and attributes can be used nearly anywhere in the syntax of the PROCEDURE DIVISION that a numeric value is acceptable, such as source operands in MOVE, ADD, and SUBTRACT statements.

Mnemonic-Name. The use of mnemonic-names provides a means of relating certain hardware equipment names to problem-oriented names the programmer may wish to use. See the discussion of SPECIAL-NAMES in Section 5.

Index-Name. An index-name is a word with at least one alphabetic character that names an index associated with a specific table (refer to indexing in Section 6). An index is a register, the contents of which represents the character position of the first character of an element of a table with respect to the beginning of the table.

Event-Name. An event-name is a word which contains at least one alphabetic character and is used as a communication link between processes. The event provides a means for interrogation between, or interlacing of, related processes. See the discussion of the USAGE IS EVENT clause in Section 6.

Lock-Name. A lock-name is a word with at least one alphabetic character that is used in an asynchronous (parallel) processing environment to name a lock A discussion of the LOCK statement is contained in Section 7.

Verbs

Another type of COBOL word is the verb. A verb is a single word that denotes an action that is to take place. These action words are used primarily in the PROCEDURE DIVISION.

Reserved Words

The third type of COBOL word is a reserved word. Reserved words are used for syntactical purposes and consist of three types:

Connectives
Optional Words
Key Words

Connectives are used to indicate the presence of a qualifier or to form compound conditions. There are three types of connectives:

a. Qualifier connectives (OF or IN) associate a data-name or a paragraph-name with its qualifier.
b. A series connective separates two or more consecutive operands. The series connective is the comma.
c. Logical connectives that are used in the formation of conditions are as follows:

AND
OR
AND NOT
OR NOT

Optional words are included in COBOL to improve readability of the statement format. These optional words may be omitted or included, as the programmer wishes; however, if an optional word is used, it must be correctly spelled.

The third kind of reserved word is the key word. A key word is a word whose presence is required when the construct in which the word appears is used in the source program. The category of key words includes the verbs previously mentioned, required words needed to complete the meaning of verb statements and entries, and words that have a specific functional meaning. A complete list of reserved words in COBOL for the B 7000/B 6000 is included in Appendix A. Reserved words can only be used in their specifically defined usage.

3. CODING FORM

GENERAL

The format of the COBOL coding form (figure 3-1) has been defined by CODASYL, by ANSI, and by common usage. The B 7000/B 6000 COBOL compiler accepts this standard format. Should program interchange be a major consideration, the user is directed to the ASA standard.

The same coding form format is used for all four divisions of a COBOL program. These divisions must appear in proper order: IDENTIFICATION, ENVIRONMENT, DATA, and PROCEDURE.

SEQUENCE FIELD (CARD COLUMNS 1-6)

The sequence field may be used to sequence the source program. Normally, a numeric sequence is used; however, the B 7000/B 6000 compiler allows any combination of characters. A warning message is given if there is a sequence error. The B 7000/B 6000 compiler provides for insertion or replacement of card images during compilation, controlled by the sequence field. (See Section 13, COBOL COMPILER.)

CONTINUATION INDICATOR (COLUMN 7)

Column 7 has several functions as follows:

a. A $ symbol in column 7 is used for cards which specify options for compiler operation. (See Section 13.)
b. If column 7 contains an asterisk (*), the rest of the card is considered to be a comment and, hence, is not "compiled" to produce object code.
c. If column 7 contains a slash (/), the listing, if any, is advanced to channel 1 and the card is considered to be a comment card.
d. The presence of a hyphen (-) indicates that the last word or literal on the previous card is not complete, but is continued on this card.

Words and numeric literals may be split at any point by placing a hyphen in column 7 of the following card. Any rightmost blank spaces on a card are ignored as are the leftmost blank spaces on the continuation card.

Burroughs COBOL CODING FORM

PROGRAM	REQUESTED BY	PAGE OF
PROGRAMMER	DATE	IDENT. 73 80

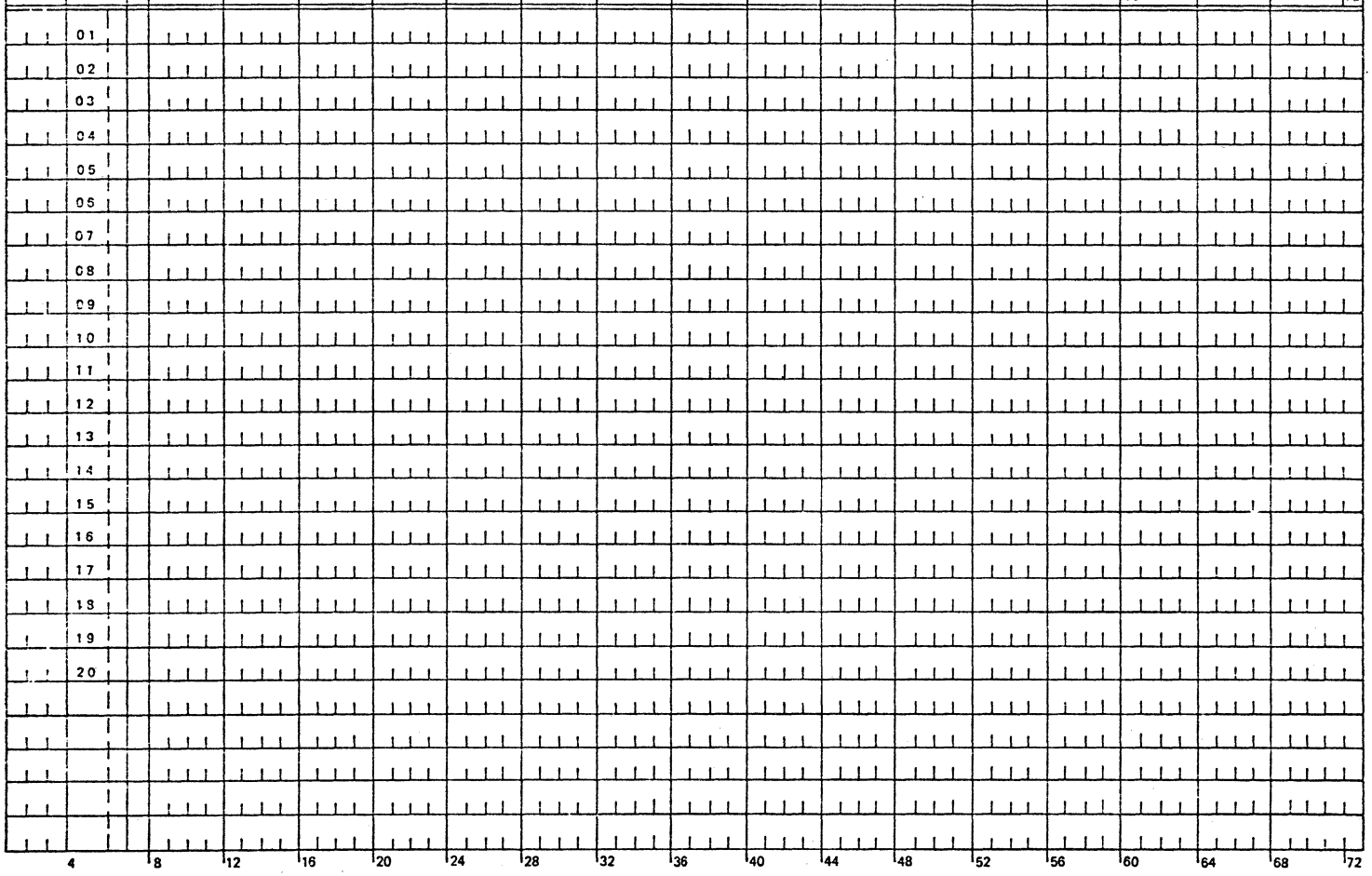


Figure 3-1. COBOL Coding Form

Non-numeric literals are split in a slightly different fashion. On the initial card, starting from the quotation mark, all information thru column 72 is taken as part of the literal, and on the next card a quote mark must be used to indicate the start of the second part of the literal.

MARGIN A (COLUMNS 8 THRU 11)

DIVISION, SECTION, and PARAGRAPH headers should always begin in margin A. A division header consists of the division name (IDENTIFICATION, ENVIRONMENT, DATA, or PROCEDURE), followed by a space, then the word DIVISION followed by a period.

A section header consists of the section-name, followed by a space and then the word SECTION, followed by an optional priority number, followed by a period. The priority number, if used, is for compatibility with other systems and is ignored unless the $ option SECGROUP is set.

A paragraph header consists of the paragraph-name followed by a period. The first sentence of the paragraph may appear on the same line as the paragraph header.

Within the IDENTIFICATION and ENVIRONMENT divisions, the section and paragraph headers are fixed and only the headers shown in this manual are permitted. Within the PROCEDURE DIVISION, the section and paragraph headers are defined by the user. Within the DATA DIVISION, level indicators should start in MARGIN A.

MARGIN B (COLUMNS 12 THRU 72)

All entries which are not DIVISION, SECTION, or PARAGRAPH headers should start in margin B.

RIGHT MARGIN (COLUMN 72)

The text of the program must appear between columns 8 and 72, inclusive. A word or statement may end in column 72.

IDENTIFICATION (COLUMNS 73 THRU 80)

The identification field may contain any information desired by the user. The field is ignored but is reproduced on the output listing by the compiler.

PUNCTUATION

The following rules of punctuation apply to the writing of COBOL programs for the B 7000/B 6000.

a. A sentence is terminated by a period followed by a space. 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 PICTURE string.

b. Two or more names in a series may be separated by a space or by a comma. If used, commas can appear only where allowed.

c. Semicolons (;) are used only for readability and are never required. If used, semicolons can appear only where allowed.

d. A space must never be embedded in an identifier (or a name). Hyphens should be used instead. (A hyphen may not start or terminate an identifier.) For example:

NET-PAY

A space, or spaces, may appear as part of a non-numeric literal.

SAMPLE CODING

An extract sample from a source program, showing the continuation of both words and non-numeric literals, is illustrated in figure 3-2.

Burroughs COBOL CODING FORM

PROGRAM CONTINUATION OF WORDS AND LITERALS | REQUESTED BY | PAGE 1 OF 1

PROGRAMMER BARBARA | DATE 19 DEC 1975 | IDENT. 73 80

```
PAGE NO. 1 3  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
01  FILE-CONTROL.                    SELECT PRINTING-FILE ASSIGN TO TAPE    RESERVE
02                                   2 ALTERNATE AREAS.
03                                   SELECT MASTER-INPUT ASSIGN TO 2 * 11500 DISK RE
04  -                                SERVE 1 ALTERNATE AREA.
05                .
06                .
07                .
08  /
09  WORKING-STORAGE SECTION.
10  01  A---FEW-LITERALS.
11      05  NUM-LIT-123                     VALUE  123
12  -           456
13  -           .78                  PICTURE  9.
14                .
15      05                           NON-NUMERIC-LITERAL                VALUE  "ALL CHARACTE
16  -       "RS  ARE VALID  .$;+-"" QUOTE"","                  SIZE 1125.
17                .
18                .
19                .
20  FIRST--PARAGRAPH.                MOVE 654321
    -       .99 TO NUM---LIT-123.
            GO      TO      FIRST-
    -       -PARAGRAPH.
```

Figure 3-2. Example of Continuation of Words and Literals

4. IDENTIFICATION DIVISION

GENERAL

The IDENTIFICATION DIVISION must be included in every COBOL source program. The structure of this division is as follows:

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

[PROGRAM-ID.    comment entry   .]
[AUTHOR.    comment entry   .]
[INSTALLATION.    comment entry   .]
[DATE-WRITTEN.    comment entry   .]
[DATE-COMPILED.    comment entry   .]
[SECURITY.    comment entry   .]
```

The IDENTIFICATION DIVISION must begin with the reserved words IDENTIFICATION DIVISION followed by a period and a space and PROGRAM-ID followed by a period and a space. IDENTIFICATION may be abbreviated as ID. All other entries in this division are optional and may appear in any sequence.

The heading and paragraph-names should begin in margin A. With the exception of the DATE-COMPILED paragraph, the entire division is copied from the input source program and listed upon the output listing.

When DATE-COMPILED is included, the first line of the entry is replaced by the current date, in the form MM/DD/YY, two digits each for month, day, and year, and the current time in the form HH:MM, AM/PM. Succeeding lines of this paragraph will be reproduced from the input source program.

CODING THE IDENTIFICATION DIVISION

Figure 4-1 provides an example of IDENTIFICATION DIVISION coding.

Burroughs COBOL CODING FORM

PROGRAM: IDENTIFICATION DIVISION CODING | REQUESTED BY: | PAGE 1 OF 1
PROGRAMMER: CHRIS | DATE 19 DEC 1975 | IDENT. 73 80

PAGE NO.	LINE NO.		A	B
	01		IDENTIFICATION DIVISION.	
	02		PROGRAM-ID.	BLISS.
	03		AUTHOR.	CHRIS AND BARBARA.
	04		INSTALLATION.	CALIFORNIA.
	05		DATE-WRITTEN.	1ST MAY 1975.
	06		DATE-COMPILED.	19 DEC 1975.
	07		SECURITY.	FOREVER.
	08	*	ALL CARDS EXCEPT DATE-COMPILED BECOME COMMENT CARDS.	
	09		THIS INCLUDES BLANK CARDS, CARDS WITH A HYPHEN IN COLUMN	
	10		-7 AND CARDS WHICH APPEAR TO HAVE A PARAGRAPH HEADER OTHER THAN	
	11		THE PARAGRAPH HEADERS WHICH ARE VALID FOR THIS DIVISION.	
	12		PURPOSE.	
	13		THIS DIVISION HAS NO EFFECT ON THE COMPILATION. EVERY	
	14		PROGRAM MUST HAVE AT LEAST THE DIVISION HEADER.	
	15			
	16			
	17			
	18			
	19			
	20			

Figure 4-1. IDENTIFICATION DIVISION Coding

5. ENVIRONMENT DIVISION

GENERAL

The ENVIRONMENT DIVISION is the second division of the source program. It is used to specify the computer being used for the compilation, to specify the computer to be used by the object program, and to specify the files to be handled by the object program. It also can be used to specify the input-output procedures to be utilized.

The ENVIRONMENT DIVISION must be included in every COBOL source program and must begin with the reserved words ENVIRONMENT DIVISION followed by a period and a space.

The general structure of the ENVIRONMENT DIVISION is as follows:

```
ENVIRONMENT DIVISION.
[CONFIGURATION SECTION.
[SOURCE-COMPUTER.   comment entry.]
[OBJECT-COMPUTER.   object-computer entry.]
[SPECIAL-NAMES.   special-names entry.]]
[{ I-O SECTION.          }
 { INPUT-OUTPUT SECTION. }
FILE-CONTROL.   file-control entry.
[I-O-CONTROL.   input-output-control entry ].]
```

The ENVIRONMENT DIVISION is comprised of two sections: the CONFIGURATION SECTION and the INPUT-OUTPUT SECTION.

The CONFIGURATION SECTION deals with the characteristics of the source computer and the object computer. The section is divided into three paragraphs: the SOURCE-COMPUTER paragraph which describes the computer configuration on which the source program is compiled, the OBJECT-COMPUTER paragraph, which describes the computer configuration on which the object program produced by the compiler is to be run, and the SPECIAL-NAMES paragraph, which relates hardware names used by the B 7000/B 6000 COBOL compiler to the mnemonic-names in the source program.

The INPUT-OUTPUT SECTION deals with the information needed to control transmission and handling of data between external media and the object program. This section is divided into two paragraphs: the FILE-CONTROL paragraph which names and associates the file with external media, and the I-O-CONTROL paragraph which defines special control techniques to be used in the object program.

The definitions of the entries for the contents of 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. The CONFIGURATION SECTION is required only when one of its paragraphs is to be used.

Source-Computer

The function of this paragraph is to describe the computer upon which the program is to be compiled.

The format of this paragraph consists of two options which are as follows:

Option 1:

```
SOURCE-COMPUTER.  COPY   library-name
                    [ FROM seq.no. ] [ { THRU    } seq.no. ]
                                       { THROUGH }
                    [ REPLACING word-1  BY  text-1
                                [,word-2  BY  text-2] ... ]
```

Option 2:

```
SOURCE-COMPUTER.    comment entry.
```

For a discussion of the COPY function, refer to Section 8, THE COBOL LIBRARY. An example of this paragraph would be:

```
SOURCE-COMPUTER. B-6700.
```

Object-Computer

The function of this paragraph is to describe the computer on which the program is to be executed and to specify core and disk size limitations when using the SORT.

The format for this paragraph consists of two options which are as follows:

Option 1:

```
OBJECT-COMPUTER.  COPY  library-name
                        [ FROM seq.no. ] [ { THRU     } seq.no. ]
                                           { THROUGH  }
                        [ REPLACING word-1 BY text-1
                                  [, word-2 BY text-2] ... ] .
```

Option 2:

```
OBJECT-COMPUTER. { B-7700 }
                 { B-6700 }
                 { word-3 }

[ , MEMORY SIZE integer { CHARACTERS } ]
                        { WORDS      }
                        { MODULES    }

[ , DISK SIZE integer { WORDS   } ]
                      { MODULES }

[ , SEGMENT-LIMIT IS integer ]

[ ,[integer] hardware-name ]... .

[ , PROGRAM COLLATING SEQUENCE IS alphabet-name ] .
```

For a discussion of the COPY function, refer to Section 8, THE COBOL LIBRARY.

Word-3 is any single COBOL word.

The MEMORY SIZE option is used only in conjunction with a SORT statement. The SORT statement may also specify MEMORY SIZE and will take precedence over the OBJECT-COMPUTER paragraph. When MEMORY SIZE is not specified in the SORT statement and not specified in the OBJECT-COMPUTER paragraph, a default MEMORY SIZE of 12,000 words will be assumed. If this option is used and a SORT statement does not appear in the program, the option will be ignored by the compiler. One module of memory is equivalent to 16,384 words of memory.

The DISK SIZE option is used only in conjunction with the SORT statement. If this clause is omitted in a sort program, DISK SIZE will be assumed to be

900,000 words. If this option is used and a SORT statement does not appear in the program, the option will be ignored by the compiler. One module of disk is equivalent to 1.8 million words of disk.

The SEGMENT-LIMIT clause can be used to control the compiler in segmenting a program. Segmentation normally occurs at the first paragraph name encountered beyond the point at which 1500 words of code have been produced and at the beginning of each SECTION of the PROCEDURE DIVISION. When SEGMENT-LIMIT is specified, the SEGMENT-LIMIT value is used instead of 1500 words. SEGMENT-LIMIT, when specified, is assumed to be in words. The maximum SEGMENT-LIMIT specification is 4095 words.

Segmentation for the initialization code of the WORKING-STORAGE SECTION will occur automatically is the SEGMENT-LIMIT has been reached.

The hardware-name list is for documentation purposes only.

If the ANSI 74 PROGRAM COLLATING SEQUENCE clause is specified, the program collating sequence is the collating sequence associated with the alphabet-name specified in that clause. This collating sequence is used to determine the truth value of any nonnumeric comparisons.

If the collating sequence clause is not specified, the native or EBCDIC collating sequence is used to determine the truth value of non-numeric comparisons.

The ANSI74 system dollar option must be set if the PROGRAM COLLATING SEQUENCE clause is to be used. Refer to Appendix B for a description of the ANSI 74 implementations.

Special-Names

The SPECIAL-NAMES paragraph relates B 7000/B 6000 hardware names to user-specified mnemonic-names and relates internal program names to external program names for tasks.

The format for the SPECIAL-NAMES paragraph consists of two options which are:

Option 1:

```
SPECIAL-NAMES.  COPY  library-name
                      [ FROM seq.no. ][ { THRU    } seq.no. ]
                                        { THROUGH }
                      [ REPLACING word-1 BY text-1
                                  [, word-2 BY text-2] ... ] .
```

Option 2:

```
SPECIAL-NAMES.
    [CURRENCY SIGN IS literal-1]
    [, DECIMAL-POINT IS COMMA]
    [, hardware-name IS mnemonic-name]...
    [, CHANNEL integer IS mnemonic-name]
    [, literal-2 IS mnemonic-name]...    .
```

For a discussion of the COPY function, refer to Section 8, THE COBOL LIBRARY.

This paragraph is required if the DECIMAL-POINT or the CURRENCY SIGN clauses are used.

Literal-1 in the CURRENCY SIGN clause is used in the PICTURE clause to represent the currency symbol. Literal-1 must be a non-numeric literal and is limited to a single character and must not be one of the following characters:

a. Digits 0 thru 9.
b. Alphabetic characters A, B, C, D, J, L, P, R, S, V, X, Z, and space.
c. Special characters * + - , . ( ) ; ".

If this clause is not present, the currency symbol ($) is used in the PICTURE clause.

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

Hardware-names may be any of the hardware-names listed at the end of the discussion of the FILE-CONTROL paragraph.

The CHANNEL integer clause is used to associate a mnemonic-name with a channel in the printer carriage control format tape. The integer may have positive values ranging from 1 thru 11. This mnemonic-name may be used only in a WRITE statement.

The last clause listed above is used to equate a mnemonic-name to an external program-name (code file title) for purposes of inter-program communication.

Literal-2 is a file-title and is in the form:

```
ID [/ID]...
```

where each ID (up to a maximum of 14) is a non-numeric literal of 1 to 17 characters in length.

Example:

```
"UP" / "TO" / "SEVENTEEN" is XYZ
```

The alphabet-name clause is an ANSI 74 addition to the existing SPECIAL-NAMES paragraph. The ANSI74 system dollar option must be set in order to make use of this feature.

Option 2 of the alphabet-name clause can be used to construct a special collating sequence in the SPECIAL-NAMES paragraph using the NATIVE (EBCDIC) character set.

The Options are as follows:

Option 1:

```
[, alphabet-name IS { STANDARD-1 }] ...
                    { NATIVE     }
                    { BCL        }
```

Option 2:

```
[                    literal-1[{THROUGH} literal-2                          ]
                              {THRU   }
, alphabet-name IS            [ALSO literal-3 [,ALSO literal-4] ...]          ]...
                    [literal-5[{THROUGH} literal-6                          ]]...
                              {THRU   }
                              [ALSO literal-7 [, ALSO literal-8] ...]       ]
```

The literals specified in Format 2 of the alphabet-name clause are implemented as follows:

a. If numeric, they must be unsigned integers and must have a value from 1 through 256.
b. If nonnumeric and associated with the THROUGH or ALSO phrase, they must be one character in length.
c. In Format 2 of the alphabet-name clause, a given character must not be specified more than once in an alphabet-name clause.

The general rules regarding the alphabet-name clause are as follows:

1. The alphabet-name clause provides a means for relating a name to a specified character code set and/or collating sequence. When alphabet-name is referenced in a PROGRAM COLLATING SEQUENCE clause, the alphabet-name clause specifies a collating sequence. When referenced in a CODE-SET clause, it specifies a character code set.
2. If the STANDARD-1 phrase is specified, the character code-set or collating sequence is ASCII.
3. If the NATIVE phrase is specified, the character code-set or collating sequence is EBCDIC.
4. **If the BCL phase is specified, the character code-set or collating sequence is BCL.**
5. **If Format 2 is specified, the alphabet-name may not be referenced in** a CODE-SET clause. The collating sequence identified is that defined according to the following rules:

Rule 1: The value of each literal specifies:

a. The ordinal number of a character within the native character set, if the literal is numeric. This value must not exceed the value which represents the number of characters in the native character set.
b. The actual character within the native character set, if the literal is non-numeric. If the value of the non-numeric literal contains multiple characters, each character in the literal, starting with the leftmost character, is assigned successive ascending positions in the collating sequence being specified.

Rule 2: Tne order in which the literals appear in the alphabet-name clause specifies, in ascending seqeunce, the ordinal number of the character within the collating sequence being specified.

Rule 3: Any characters within the native collating sequence, which are not explicitly specified in Format 2, assume a position in the collating sequence being specified, greater than any of the explicitly specified

characters. The relative order within the set of these unspecified characters is unchanged from the native collating sequence.

Rule 4: If the THROUGH phrase is specified, the set of contiguous characters in the native character set beginning with the character specified by the value of literal-1, ending with the character specified by the value of literal-2, is assigned a successive ascending position in the collating sequence being specified. In addition, the set of continguous characters may be in either ascending or descending order.

Rule 5: If the ALSO phrase is specified, the characters of the native character set specified by the value of literal-1, literal-3, literal-4,..., are assigned to the same position in the collating sequence being specified.

Rule 6: The character that has the highest ordinal position in the program collating sequence specified is associated with the figurative constant HIGH-VALUE.

Rule 7: The character that has the lowest ordinal position in the program collating sequence specified is associated with the figurative constant LOW-VALUE.

Refer to Appendix B for a description of the ANSI 74 implementations.

INPUT-OUTPUT SECTION

The INPUT-OUTPUT SECTION contains information concerning files to be used in the object program, the manner of recording used or to be used, special rerun points, and the presence of any multiple-file tape.

FILE-CONTROL

The FILE-CONTROL paragraph names each file, identifies the file medium, and allows particular hardware assignments. This paragraph also specifies alternative input-output areas. The format for FILE-CONTROL consists of two options which are:

Option 1:

```
FILE-CONTROL.  COPY library-name
                    [FROM seq.no.][{THRU    } seq. no.]
                                  {THROUGH}
                    [REPLACING word-1 BY text-1
                               [,word-2 BY text-2]...].
```

Option 2:

```
FILE-CONTROL.
   SELECT [LOCAL ] [RECEIVED BY {REFERENCE}] [OPTIONAL]  file-name
          [GLOBAL]              {REF      }

       ASSIGN TO integer-1 [* integer-2] [INTERCHANGE]
          [DIRECT]  hardware-name-1  [, hardware-name-2] ...
             [OR  hardware-name-3  [, hardware-name-4] ...]
          [SINGLE]  [BY {AREA    }]    [SAVE]
                        {CYLINDER}

       [;RESERVE  {NO        }  [ALTERNATE  [AREA ]]]
                  {integer-3 }              [AREAS]
                  {data-name-1}
       [; {FILE-LIMIT IS  } {data-name-2}{THROUGH} {data-name-3}]
          {FILE-LIMITS ARE} {literal-1  }{THRU   } {literal-2  }
                                                   {END       }
       [;  ACCESS MODE IS {SEQUENTIAL}]
                          {RANDOM    }
       [;  ACTUAL KEY IS data-name-4].
       [;  ORGANIZATION IS SEQUENTIAL]
       [;  FILE STATUS IS data-name-5].
```

With the exception of the ASSIGN clause which must follow the SELECT clause, the clauses may appear in any order.

Each file described in the DATA DIVISION must be named once and only once as file-name in the FILE-CONTROL paragraph. Each file specified in the SELECT clause must have a File Description entry or a Sort-Merge File Description entry in the DATA DIVISION.

COPY Function

For a discussion of the COPY function refer to Section 8, THE COBOL LIBRARY.

SELECT Clause

Each file described in the DATA DIVISION must be named once and only once as a file-name in the FILE-CONTROL paragraph following the key word SELECT. Each selected file must have a file description entry in the DATA DIVISION. Hyphens should not be imbedded in file-name so as to avoid operational problems of the MCP. Selection of file-names should be done judiciously since these names must be used when control cards are needed. Although the compiler allows file-names to be up to 30 characters in length, the MCP will allow up to 17 characters and truncate any excess. File-names containing embedded hyphens or having a digit as the first character of the name, must be enclosed in quotes in MCP messages and control cards.

If LOCAL is used, then the file is a formal parameter for a procedure and may only be named in WITH and USING clauses in the declarative USE statement associated with this procedure.

The OPTIONAL clause may only be specified for input files. Its specification is required for input files that are not necessarily present each time the object program is executed.

Files and direct files may be declared GLOBAL by an extension to the syntax of the SELECT clause.

Example:

```
SELECT GLOBAL FYLE ASSIGN TO DIRECT DISK.
```

There is a restriction for non-direct files when the GLOBAL clause is used. The first record description must match a similar record description for the file in the host, by name and array type. (The $ option GLOBAL has no effect on ENVIRONMENT DIVISION or FILE SECTION entries.)

The ASSIGN clause must be used to associate an input or output file with the proper hardware device. The full USASI-1968 and ANSI-1974 syntax for "Hardware-name" is recognized by this compiler. "Hardware-name-1" is used by the compiler for device assignment. The remaining hardware-names are ignored. Except in the case of sort files and disk files, integer-1 is optional and used for documentation only. (Refer to the B 7000/B 6000 System Software Operational Guide, Volume 1, Form No. 5001563, for information concerning SORT files.)

If a file is assigned to integer-1 SORT-TAPES, then integer-1 must be between 3 and 8, inclusive, and the RESERVE integer-3 ALTERNATE AREAS option is prohibited unless integer-3 equals 1.

If a file is assigned to SORT DISK AND integer-1 SORT-TAPES, then integer-1 must be between 3 and 8, inclusive.

DIRECT hardware-name is a B 7000/B 6000 extension which provides the user programmatic control of input-output operations and the ability to use the same input or output file in two or more asynchronously processed programs.

When DIRECT hardware-name is used, no input-output buffers are assigned and the physical records of the file are read directly into or written directly from the WORKING-STORAGE area provided by the programmer. The RESERVE ... and FILE-LIMIT(S) ... clauses are not used on DIRECT files.

The ASSIGN TO integer-1 DISK is used to designate the maximum number of logical records to be placed in a file on disk. It must be specified only for files which are to be created on disk. For the integer-1 * integer-2 DISK option, a file may be thought to contain partitions or areas. Each area is independently allocated when actually referenced. For example, a file which occasionally contains 10,000 records, but normally only 1000, may be divided into 10 areas, each with 1000 records by stating: ASSIGN TO 10 * 1000 DISK.

When only one integer is specified, the file will consist of a single area of "integer" size. When neither of the integers are specified, the MCP will make the assignment based on the values available for an existing file of the same name at execution time and, in fact, use the disk area and the header of the existing file. If no file exists with the identical title, at execution time, then the value 20 * 1000 is assigned by the MCP.

When this "area size" option is used, the maximum value permitted for integer-1 is 1000; for integer-2, 1,048,575.

The SAVE option, when used with the SELECT clause, provides system compatibility with B 3700 COBOL. This feature is available only while the B2500 system option is set. Refer to Appendix C for additional information concerning the B 2500 system dollar option.

Using the SAVE option in the SELECT clause causes the PROTECTION attribute to be given the value protected in the file description.

RESERVE Clause

The RESERVE option allows an additional number of input or output buffers to be supplied for file-name. Two buffers are automatically supplied when the option is omitted. If NO is used, then only one buffer is reserved (minimum specification). When a RESERVE clause is present, the integer value of integer-3 or data-name-1 determines the number of buffers supplied, in addition to the minimum number of one. Data-name-1 is used for dynamic allocation of buffers and must be defined as an elementary numeric item.

For object program efficiency, do not reserve more alternate areas than are needed in the program. Also, do not specify NO ALTERNATE AREAS for a file unless the references to the files are very infrequent. The advantage of two and only two output areas being supplied automatically (in the absence of the RESERVE option) is increased efficiency in the object program.

Disk File Options

The FILE-LIMIT, ACCESS MODE, and ACTUAL KEY clauses apply to disk files and dispack files. (The ACCESS MODE and ACTUAL KEY clauses are also used with data communications files, but their meanings are slightly different. All programming considerations for data communications files are discussed in Section 10.

FILE-LIMIT Clause

In the FILE-LIMIT clause, the pair of operands associated with the key word THRU represents a logical segment of the file. The values of data-name-2 and data-name-3 or literal-1 and literal-2 correspond to the low and high values of the ACTUAL KEY of records available to the program; that is, the logical beginning of a disk file is considered to be that address represented by literal-1 or data-name-2. The operand END specifies that the end of the disk file is to be the last record written before use of the file as input.

The value of the data items as specified in the FILE-LIMIT clause is utilized by the system only at the time that the associated disk file is opened by the execution of the OPEN statement.

In order to ensure that FILE-LIMITS will work correctly, the file must be opened explicitly by an OPEN statement. If data-names are declared to be file-limits, the value of those items at the time of the OPEN statement determines the limits of the file while it is open. Opening the file by setting the attribute MYUSE and OPEN may cause unpredictable AT END or INVALID KEY branching to be taken for subsequent READ or WRITE statements.

FILE-LIMITS may be used on either RANDOM or SEQUENTIAL input, output, and I/O files.

ACCESS Clause

When ACCESS MODE SEQUENTIAL is specified, records are handled sequentially. The next record to be read or written is that contiguous to the current record. No ACTUAL KEY entry is necessary for the SEQUENTIAL mode.

If the ACCESS MODE RANDOM clause is specified, the ACTUAL KEY entry must also be specified. In this case, the MCP obtains each record randomly; the next record to be read or written is that record addressed by the ACTUAL KEY entry.

When the ACCESS clause is not specified, ACCESS SEQUENTIAL is implied.

ACTUAL KEY Clause

The ACTUAL KEY clause is optional for sequential access files. When a SEQUENTIAL access file specifies an ACTUAL KEY, data-name-4 will be updated to contain the relative record number of the current record. The current record is the most recently read or written data record. Sequential access files opened input or output increment the value of actual key by 1 for each READ or WRITE. Sequential access files opened input-output, also increment by 1, except when a record is updated with a WRITE following a READ; thus, properly reflecting the current record number at all times.

For sequential files whose ACTUAL KEY is specified:

a. The MCP will always copy its internal key into the ACTUAL KEY, except for DIRECT files which must use a direct area attribute.

b. Programmatic changes of the ACTUAL KEY will not alter the access sequence of a serial file unless a SEEK is explicitly given after altering the ACTUAL KEY and prior to a READ or WRITE. For example:

```
MOVE 80 TO ACT-KEY.
SEEK MASTER-FILE.
```

The above statements show how to set an ACTUAL KEY to point to a particular record and then resume processing at record #80.

c. When a SEEK statement is used to start the accessing of the sequential file at a particular point within the file, the contents of ACTUAL KEY cannot contradict the FILE-LIMITS restriction (i.e., the ACTUAL KEY value must be within the specified bounds of the file). If the ACTUAL KEY value is not within the specified bounds, an INVALID KEY error occurs when the file is accessed.

If the ACCESS MODE RANDOM clause is specified, the ACTUAL KEY entry must contain the relative record-number of the record to be read or written. The contents of data-name-4 are used by the SEEK statement (or, in its absence, the READ and WRITE statements) to locate a specific disk record. Therefore, the location (address) must have been placed in data-name-4 prior to the execution of a SEEK, READ, or WRITE statement.

Values of data-name-4 are controlled by the programmer. The value may range from 1 to N, where N equals the number of records on the file or as limited by the FILE-LIMITS clause. The ACTUAL KEY gives the relative position of the record within the file. For optimum results, the ACTUAL KEY should be a data-name that is defined as a non-contiguous item (77 level) within the WORKING-STORAGE SECTION and should have a PICTURE of 9(11) and be COMPUTATIONAL-1. In all cases, data-name must be a numeric integer.

ORGANIZATION/FILE STATUS Clauses

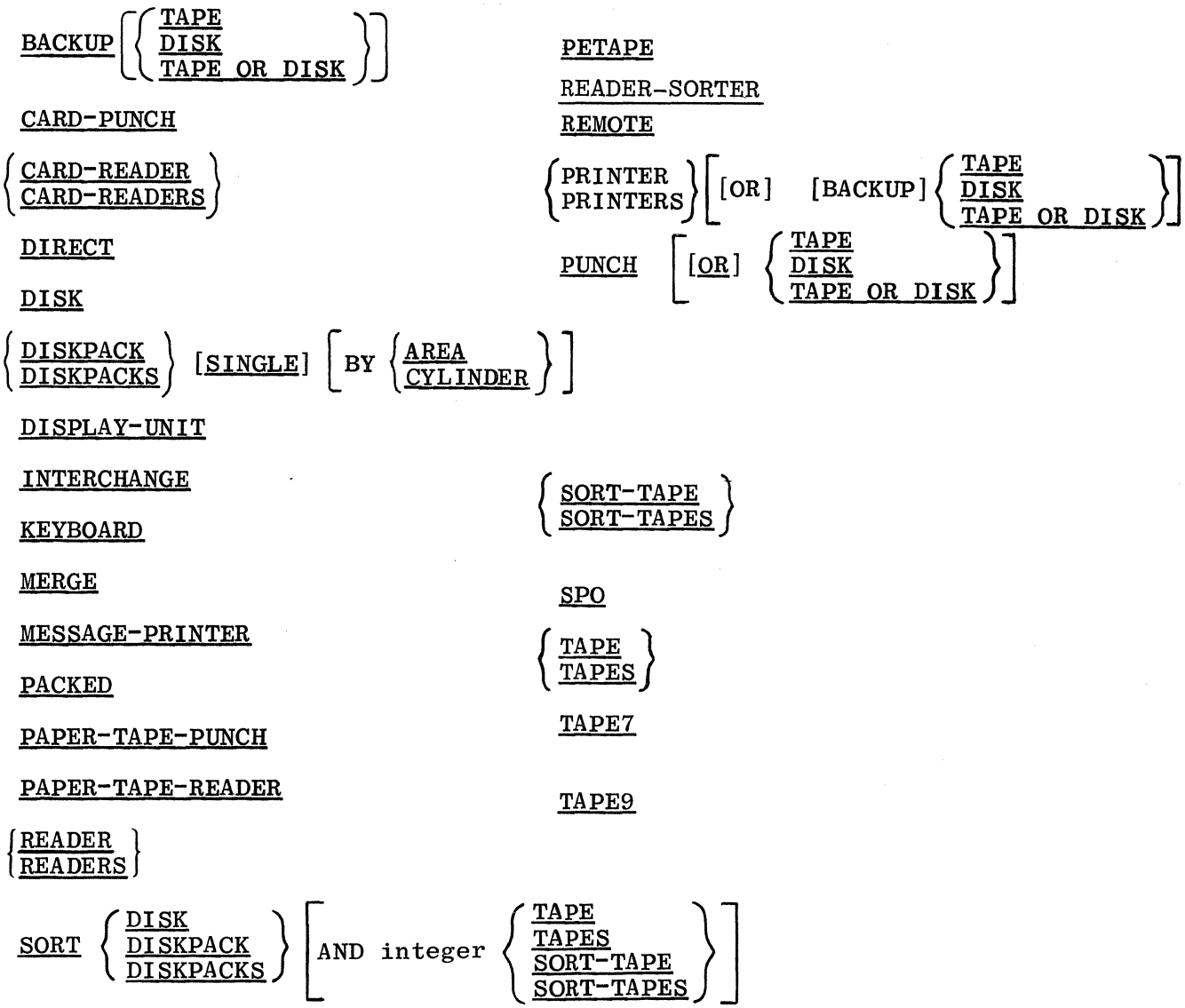
When the ANSI74 system dollar option is set, the ORGANIZATION and FILE STATUS clauses for sequential I/O files can be specified.

If the ORGANIZATION clause is omitted, sequential organization is assumed. The FILE STATUS data-name-5 must be declared as a two-character data item of category alphanumeric. Following the execution of all I/O statements, the appropriate value is stored in the FILE STATUS data item. The following table shows the values stored into the FILE STATUS data item and their corresponding meanings:

"00"	SUCCESSFUL COMPLETION
"10"	END OF FILE
"30"	PARITY ERROR
"91"	SHORT BLOCK
"92"	DATA ERROR
"96"	BREAK ON OUTPUT
"97"	SECURITY VIOLATION
"98"	I-O TIME LIMIT
"99"	UNEXPECTED I-O ERROR

Refer to Appendix B for a description of the ANSI 74 implementations.

The allowable entries for hardware-name are as follows:



When TAPE or TAPES is specified, an assignment is made to the first available tape unit encountered. PETAPE specifies phase-encoded tape unit.

BACKUP infers PRINTER unless PUNCH is specified.

I-O-CONTROL

The I-O-CONTROL paragraph specifies blocking techniques, file location on multiple file reels, and shared memory areas.

The format for this paragraph consists of the following two options:

Option 1:

```
I-O-CONTROL.  COPY library-name
                    [FROM seq. no.][{THRU / THROUGH}   seq. no.]
                    [REPLACING word-1 BY text-1
                              [, word-2 BY text-2]  ... ] .
```

Option 2:

```
I-O-CONTROL.
[APPLY comment entry] ...
[; MULTIPLE FILE TAPE CONTAINS [file-name-1] [POSITION integer-1] ...]...
[; SAME [RECORD / SORT] AREA FOR file-name-2 {,file-name-3}... ] ...
[; RERUN ON {DISK / DISKPACK}  EVERY integer-2 RECORDS OF file-name-4  ...].
```

For a discussion of the COPY function, refer to Section 8, the COBOL LIBRARY.

The I-O-CONTROL paragraph may be omitted in the absence of any clause entries.

The APPLY clause is for documentation only. Its intent is to indicate the blocking format of the file. (See RECORD CONCEPTS in Section 6.) The comment entry can contain any text except the words MULTIPLE, SAME, RERUN, or the character period.

More than one file may be read from or written to a single reel by listing these files in the MULTIPLE FILE TAPE CONTAINS clause in the I-O-CONTROL paragraph of the INPUT-OUTPUT SECTION.

Use one clause for each multiple file tape. The titles of all the files listed in a given clause should have a common volume-ID.

After each file is read or written, CLOSE file-name WITH NO REWIND and execute an OPEN file-name WITH NO REWIND for the next file. If the volume-ID is correct, it will be written to or read from the same reel.

The use of SAME RECORD AREA is implemented for non-DIRECT files having a common internal mode and maximum record size.

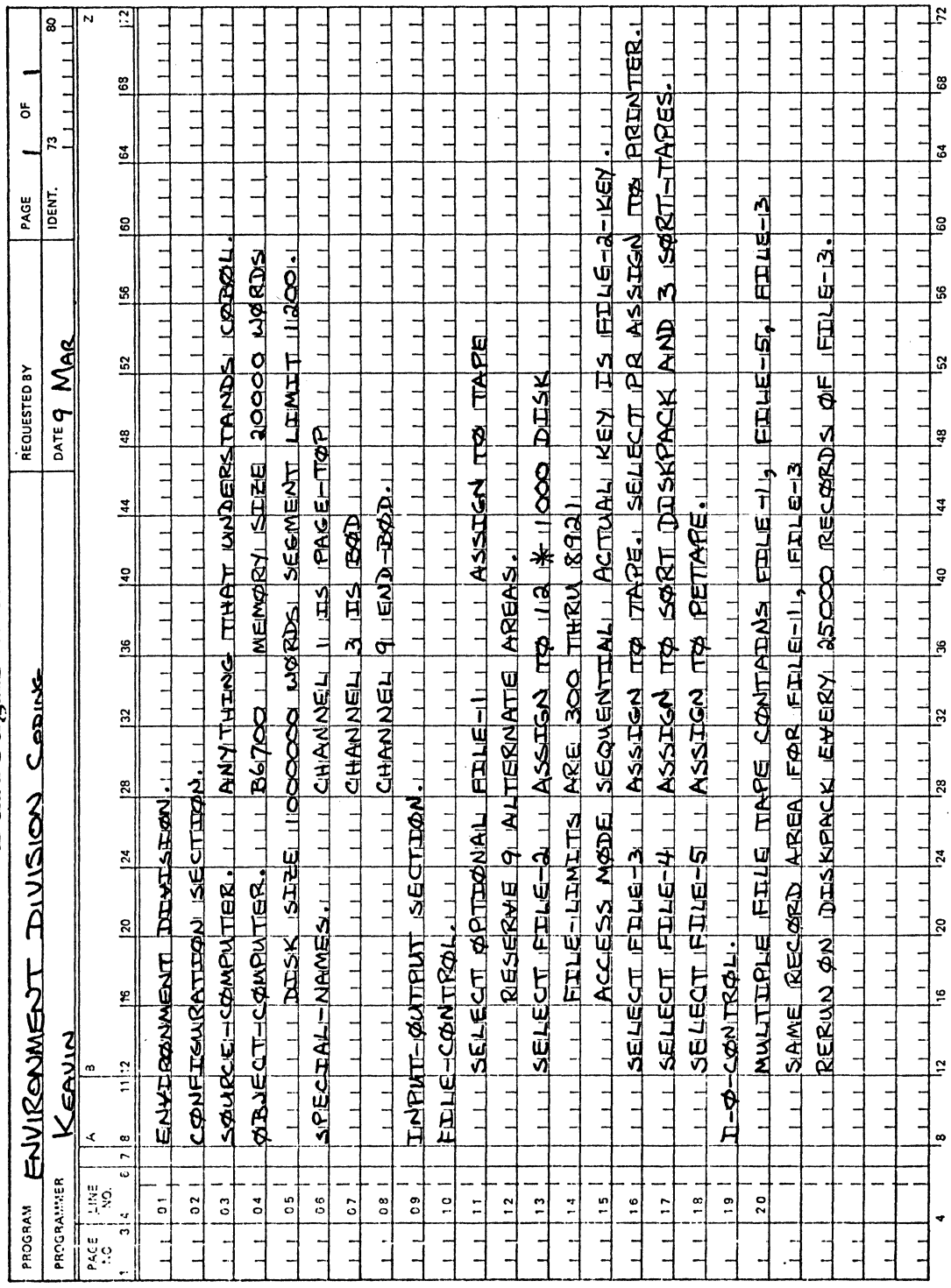
The internal mode for each file is obtained from the usage of the first record description for that file. The maximum record size for each file is obtained from the maximum length record size declared.

The SAME AREA and SAME SORT AREA clauses are used for documentation purposes only, as the operating system assigns memory for buffers only when the file is open.

The RERUN specifies that after reading integer-2 records from (or writing integer-2 records to) file-name-4 a check-point will be written to disk or diskpack so that the program may be restarted at a selected point.

CODING THE ENVIRONMENT DIVISION

Figure 5-1 illustrates the manner in which the ENVIRONMENT DIVISION is coded.



Burroughs COBOL CODING FORM

PROGRAM: ENVIRONMENT DIVISION Coding — REQUESTED BY — PAGE 1 OF 1

PROGRAMMER: KEAVIN — DATE: 9 MAR — IDENT.

```
01  ENVIRONMENT DIVISION.
02  CONFIGURATION SECTION.
03  SOURCE-COMPUTER.    ANYTHING THAT UNDERSTANDS COBOL.
04  OBJECT-COMPUTER.    B6700    MEMORY SIZE 20000 WORDS
05                       DISK SIZE 10000000 WORDS, SEGMENT LIMIT 11200.
06  SPECIAL-NAMES.      CHANNEL 1 IS PAGE-TOP
07                       CHANNEL 3 IS BAD
08                       CHANNEL 9 IS END-BOD.
09  INPUT-OUTPUT SECTION.
10  FILE-CONTROL.
11      SELECT OPTIONAL FILE-1    ASSIGN TO TAPE
12          RESERVE 9 ALTERNATE AREAS.
13      SELECT FILE-2    ASSIGN TO 112 * 1000 DISK
14          FILE-LIMITS ARE 300 THRU 8921
15          ACCESS MODE SEQUENTIAL    ACTUAL KEY IS FILE-2-KEY.
16      SELECT FILE-3    ASSIGN TO TAPE.    SELECT OR ASSIGN TO PRINTER.
17      SELECT FILE-4    ASSIGN TO SORT DISKPACK AND 3 SORT-TAPES.
18      SELECT FILE-5    ASSIGN TO PETAPE.
19  I-O-CONTROL.
20      MULTIPLE FILE TAPE CONTAINS FILE-1, FILE-5, FILE-3
        SAME RECORD AREA FOR FILE-1, FILE-3
        RERUN ON DISKPACK EVERY 25000 RECORDS OF FILE-3.
```

Figure 5-1. ENVIORNMENT DIVISION Coding

6. DATA DIVISION

GENERAL

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

a. That which is contained in files and enters or leaves the internal memory of the computer from areas of the FILE, REPORT, DATA-BASE, or LOCAL-STORAGE SECTION.
b. That which is developed internally and placed into intermediate areas of the WORKING-STORAGE SECTION.
c. Constants which are defined by the user in the CONSTANT SECTION.

The DATA DIVISION begins with the reserved words DATA DIVISION followed by a period. Immediately after is the provision for the specification of a PREPARED FOR clause:

DATA DIVISION. [PREPARED FOR system-name.]

The PREPARED FOR clause appears on the same line as the heading DATA DIVISION, separated by one or more spaces. The clause consists of the reserved words PREPARED FOR followed by any system-name and/or comment entry. This clause is optional and is used for documentation only.

The following gives the general format of the sections in the DATA DIVISION, and defines the order of their presentation in the source program.

```
DATA DIVISION.   [PREPARED FOR system-name.]

[FILE SECTION.

     [{file-description-entry          } [record-description-entry] ...] ...]
      {sort-merge-description-entry    }

[DATA-BASE SECTION.

    [01 [internal-set-name] INVOKE set-name]...]

[WORKING-STORAGE SECTION.

    [{77-level-description-entry  } ...]
     {record-description-entry    }

[CONSTANT SECTION.

    [{77-level-description-entry  } ...]
     {record-description-entry    }

[LINKAGE SECTION.

    [{77-level-description-entry  } ...]
     {record-description-entry    }

[LOCAL-STORAGE SECTION.

    [ LD local-storage-name.

        [{77-level-description-entry  } ...] ...]
         {record-description-entry    }

[REPORT SECTION.

    [report-description-entry  {report-group-description-entry} ...] ...]
```

Each section of the DATA DIVISION is optional and may be omitted from the source program if not needed. However, if a section is included, it must be incorporated in order of appearance shown above. These sections are described on the following pages.

The file description defines information pertaining to the physical aspects of a file. Such items as number of records in a block, identification of records in the file, the presence or absence of labels, etc., are included to describe the entire file.

The record description presents logical characteristics of each record. This includes the layout of items within each record type, size of various items in the record, indication of the range of values for each item, picture of the contents of each item, whether the item is signed or not, and the usage of an item within the program. All of these parameters may be utilized to define logical characteristics of each record.

The WORKING-STORAGE, CONSTANT, LINKAGE, and LOCAL-STORAGE SECTIONS are comprised of internal record descriptions and individual unrelated items, which are described as record entries, or parts of record entries.

The REPORT SECTION is described in section 11.

In summary, the DATA DIVISION contains information pertaining to the data to be used by the program: the files used, the records contained in each file, and items comprising each record; in addition, working storage and constants may be specified.

FILE AND RECORD CONCEPTS

The approach taken in defining file information is to distinguish between the physical aspects of the file and the conceptual characteristics of the data contained within the file.

Physical Aspects of a File

The physical aspects of a file describe the data as it appears on the input or output media and include such features as the following:

a. The mode in which the data file is recorded on the external medium.
b. The grouping of logical records within the physical limitations of the file medium.
c. The means by which the file can be identified.

Conceptual Characteristics of a File

The conceptual characteristics of a file explicitly define each logical entity within the file itself. In a COBOL program, the input or output statements refer to one logical record.

It is important to distinguish between a physical record and a logical record. For COBOL a logical record is a group of related information, uniquely identifiable, that is treated as a unit.

A physical record is a physical unit of information whose size and recording mode are convenient to a particular computer for the storage of data on an input or output device. The size of a physical record is hardware-dependent and bears no direct relationship to the size of the file of information contained on a device.

A logical record may be contained within a single physical unit; or several logical records may be contained within a single physical unit; or a logical record may require more than one physical unit to contain it. There are several source-language methods available for describing the relationship of logical records and physical units. Once the relationship has been established, the control of the accessibility of logical records as related to the physical unit is the responsibility of the operating system. In this manual, reference to records means to logical records, unless the term "physical record" is specifically used.

The concept of a logical record is not restricted to files but may be applied to all sections of the DATA DIVISION.

Record Concepts

The record description consists of a set of DATA DESCRIPTION entries which describe the characteristics of a particular record. Each DATA DESCRIPTION entry consists of a level-number followed by a data-name, followed by a series of independent clauses, as required.

Example:

```
01 ITEM-ONE SIZE 6.
```

The maximum size of a record description (i.e., the sum of the maximum sizes of all the items subordinate to an 01 level item) is restricted by the explicit or implicit USAGE of the 01 item. For USAGE of DISPLAY, DISPLAY-1 or ASCII, the maximum size is 65,535 characters. For USAGE of COMP, COMP-1, COMP-4, COMP-5, INDEX, EVENT, LOCK or CP, the maximum size is 65,535 words. For USAGE of COMP-2, the maximum size is 65,535 digits.

LEVEL NUMBERS CONCEPT

The concept of hierarchy is inherent in the structure of a logical record. This concept arises from the need to specify subdivisions of a record for the purpose of data reference. Once a subdivision has been specified, it may be further subdivided to permit more detailed data referral. In other words, level numbers define the interrelationship of the items comprising the record and allow the programmer to access individual items or groups of items.

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

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

In COBOL, the item relationship is specified by the use of a series of level numbers. These numbers may range from 1 thru 49. (Special level numbers of 66, 77, and 88 are discussed later.)

Each record of a file begins with the level number 1 (which may also be written as 01). This number is reserved for the record name only, as the most generic grouping. Less inclusive groupings are given higher numbers (not necessarily successive) up to a limit of 49. Figure 6-1 illustrates a form of level construction.

The smallest elements of the description are called elementary items. In figure 6-1, EMP-NO, EMP-COST-CENTER, EMP-LAST-NAME, EMP-FIRST-NAME, and EMP-M-INITIAL are all elementary items, as well as EMP-ANNUAL-SALARY, EMP-MONTH, EMP-HDAY, EMP-HYEAR, EMP-GROSS, EMP-HOSPITAL, EMP-LIFE, EMP-FICA, EMP-STATE-TAX, EMP-WITHHOLDING, EMP-LMONTH and EMP-LDAY. None of these items are further subdivided; therefore they are called elementary items.

Each elementary item belongs to one or more groups. In the example, EMP-HOSPITAL is a part of the EMP-INSURANCE group. EMP-INSURANCE, in turn, is part of the EMP-DEDUCTIONS group, which is part of the EMP-PAY-DATA group. Therefore, a group is defined as being composed of all group and elementary items described under it, until a level number equal to or less than the

Burroughs COBOL CODING FORM

PROGRAM: LEVEL NUMBER CONSTRUCTION | REQUESTED BY | PAGE 1 OF 1
PROGRAMMER: DARCY | DATE 21 SEPT | IDENT. 73 80

```
      01  01  EMPLOYEE-INFO.
      02      03  EMP-NO                          PIC 9(15).
      03      03  EMP-COST-CENTER                 PIC 9999.
      04      03  EMP-NAME.
      05          05  EMP-LAST-NAME               SIZE 13.
      06          05  EMP-FIRST-INITIAL           SIZE 1.
      07          05  EMP-M-INITIAL               PIC X.
      08      03  EMP-ANNUAL-SALARY               PIC 9(6)V99.
      09      03  EMP-DT-HIRED                    SIZE 6.
      10          05  EMP-HMONTH          PIC 99.
      11          05  EMP-HDAY            PIC 99.
      12          05  EMP-HYEAR           PIC 99.
      13      03  EMP-PAY-DATA.
      14          05  EMP-GROSS                   PIC 9(6)V99.
      15          05  EMP-DEDUCTIONS.
      16              09  EMP-INSURANCE.
      17                  11  EMP-HOSPITAL        PIC 9999V99.
      18                  11  EMP-LIFE            PIC 9999V99.
      19              09  EMP-TAXES.
      20                  11  EMP-FICA            PIC 9999V99.
                          11  EMP-STATE-TAX       PIC 9(4)V9(2).
                          11  EMP-WITHHOLDING     PIC 9(4)V99.
      03      03  EMP-LAST-REVIEW.
                  04  EMP-LMONTH                  PIC 99.
                  04  EMP-LDAY                    PIC 99.
```

Figure 6-1. Level Number Construction

group level number is encountered. In the example, EMP-PAY-DATA group includes all items to, but not including, EMP-LAST-REVIEW (which has an equal level number). Likewise, EMP-DEDUCTIONS group includes all subsequent items up to, but not including, EMP-LAST-REVIEW (which has a level number less than EMP-DEDUCTIONS).

Level numbers used in defining successively smaller groupings, working toward an elementary item, are given in larger values. Although it is not necessary that they be consistent or consecutive, a level number must not exceed 49.

A level number immediately following the last elementary item of a group must have a value of less than or equal to the level number for that group and equal to the level number of some previous group. An exception is that level number 1 (or 01) is reserved exclusively for identifying the beginning of a record description.

In the above example, the rule prohibits EMP-ANNUAL-SALARY from having a level number of 2 (or 02). Likewise, the entry name EMP-LAST-REVIEW could not have had a level number of 10 or 06 because, in the example, no previous group appears with either of these levels. As a completely separate group, it could only have a level number the same as that of the major groups previously shown.

Figure 6-2 illustrates another way to visualize the concept of level numbers by using the same example.

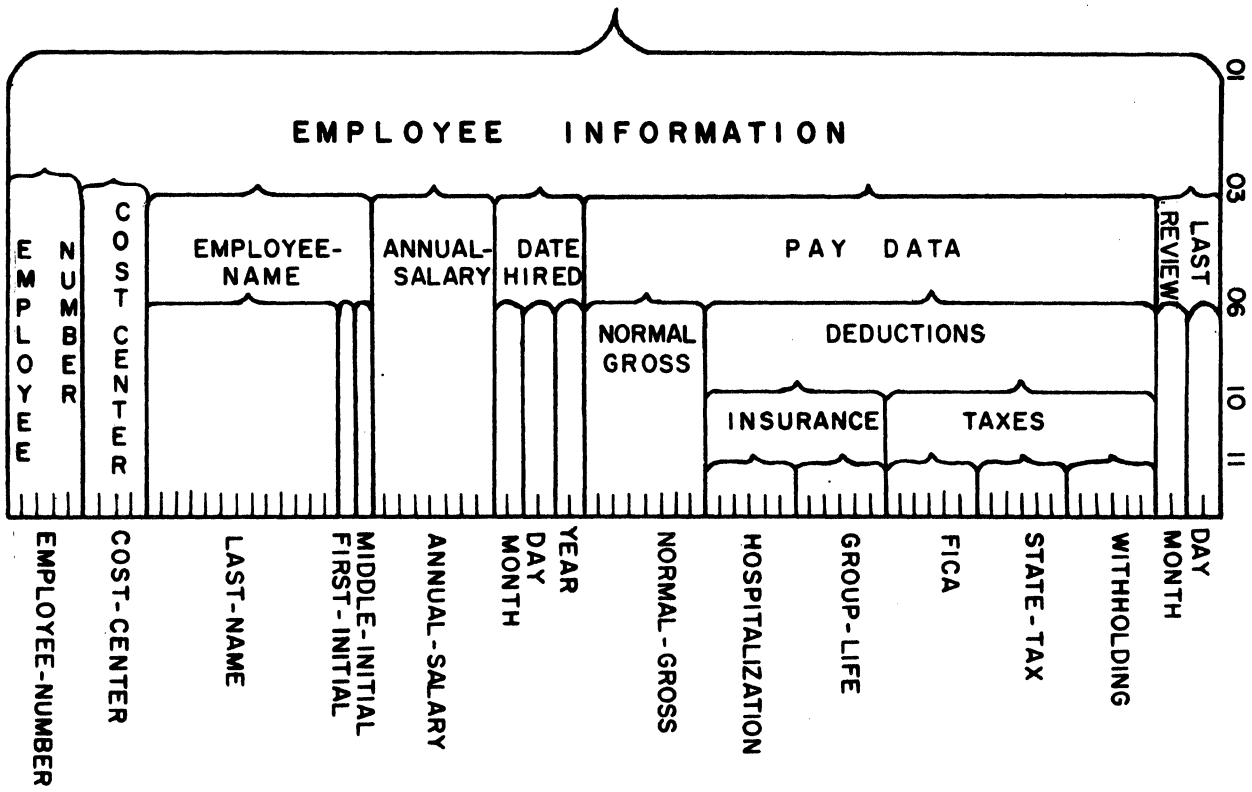


Figure 6-2. Concept of Level Numbers

QUALIFICATION

Every user-defined name explicitly referenced in a COBOL source program must be uniquely referenced either because no other name has the identical spelling and hyphenation or because it is unique within the context of a REDEFINES clause, or because the name exists within a hierarchy of names such that reference to the name can be made unique by mentioning one or more of the higher-level names in the hierarchy. These higher-level names are called qualifiers and this process that specifies uniqueness is called qualification. Identical user-defined names may appear in a source program; however, uniqueness must then be established through qualification for each user-defined name explicitly referenced, except in the case of redefinition. All available qualifiers need not be specified so long as uniqueness is established. Reserved words naming the special registers require qualification to provide uniqueness of reference whenever a source program would result in more than one occurrence of any of these special registers.

The hierarchy of qualification is as follows: names associated with a level indicator are the most significant; then names associated with level-number 01, then those names associated with level-number 02, ... , 49. A section-name is the highest (and the only) qualifier available for a paragraph-name. Thus, the most significant name in the hierarchy must be unique and cannot be qualified. Subscripted or indexed data-names and conditional variables, as well as paragraph-names and data-names, may be made unique by qualification. The name of a conditional variable can be used as a qualifier for any of its condition-names.

Regardless of the available qualification, no name can be both a data-name and a procedure-name.

Qualification is performed by following a data-name or a paragraph-name by one or more phrases composed of a qualifier preceded by IN or OF. IN and OF are logically equivalent.

The format for qualification consists of five options which are shown below:

Option 1:

```
{ data-name-1    }  { { { IN } data-name-2 } ... [ { IN } file-name ] }
{ condition-name }  { { { OF }             }     [ { OF }           ] }
                    {                                                 }
                    { { IN } file-name                                }
                    { { OF }                                          }
```

Option 2:

```
paragraph-name { OF } section-name
               { IN }
```

Option 3:

```
LINAGE-COUNTER { IN } file-name
               { OF }
```

Option 4:

```
{ PAGE-COUNTER } { IN } report-name
{ LINE-COUNTER } { OF }
```

Option 5:

```
data-name-3 { { IN } data-name-4 [ { IN } report-name ] }
            { { OF }             [ { OF }             ] }
            {                                           }
            { { IN } report-name                        }
            { { OF }                                    }
```

QUALIFICATION

The rules for qualification are as follows:

a. Each qualifier must be of a successively higher level and within the same hierarchy as the name it qualifies.
b. The same name must not appear at two levels in a hierarchy so that the name would appear to qualify itself.
c. If a data-name or a condition-name is assigned to more than one data item in a source program, the data-name or condition-name must be qualified each time it is referred to in the PROCEDURE DIVISION, ENVIRONMENT DIVISION, and DATA DIVISION (except REDEFINES where, by definition, qualification is unnecessary).
d. A paragraph-name must not be duplicated within a section. When a paragraph-name is qualified by a section-name, the word SECTION must not appear. A paragraph-name need not be qualified when referenced within its own section.
e. A data-name cannot be subscripted or indexed when it is being used as a qualifier.
f. A name can be qualified, even though it does not need qualification; if there is more than one combination of qualifiers that ensures uniqueness, then any such set can be used.
g. LINAGE-COUNTER must be qualified each time it is referenced if more than one File Description entry containing a LINAGE clause has been specified in the source program.
h. PAGE-COUNTER and LINE-COUNTER must be qualified each time they are referenced if more than one Report Description entry has been specified in the source program.

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 one PREFIX into the other PREFIX, the PROCEDURE DIVISION must be coded with one of the following sentences:

a. MOVE PREFIX IN ITEM-NO TO PREFIX OF 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 IN 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 that appears in a table (i.e., array, list, etc.). For example, a master record might contain 16 total fields, and these might be described as TOTAL-ONE, TOTAL-TWO, etc. However, this requires 16 data-names, and each total must be individually referenced in the PROCEDURE DIVISION. A more powerful way to describe the field is:

```
TOTAL . . . OCCURS 16 TIMES.
```

Elements of a table are referenced thru the use of subscripting or indexing. An element of a table is represented by an occurrence number.

The elements of a table may contain subordinate fields. For example:

```
02 TOTAL SIZE 24 . . . OCCURS 16 TIMES.
    03 TOTAL-A . . . PICTURE 9(6).
    03 TOTAL-B . . . PICTURE 9(6) OCCURS 3 TIMES.
```

Also, as shown above, OCCURS may be nested to describe tables of more than one dimension by applying an OCCURS clause to a subordinate name. Standard COBOL limits tables to three-dimensions; however, because of the unique hardware features of the B 7000/B 6000, OCCURS may be nested to any desired depth up to the limit of 49 imposed by the range of level numbers. Figure 6-3 shows an example of a multi-dimensioned table.

In the WORKING-STORAGE and CONSTANT SECTIONS, initial values of elements within tables are specified in one of the following ways:

a. The table may be described as a record by a set of contiguous data description entries, each of which specifies the VALUE of an element, or part of an element, of the table. In defining the record and its elements, any data description clause (SIZE, USAGE, PICTURE, etc.) may be used to complete the definition, where required. This form is required when the elements of the table require separate handling due to synchronization, USAGE, etc. The hierarchical structure of the table is then shown by use of the REDEFINES entry and its associated subordinate entries. The subordinate entries following the REDEFINES entry, which are repeated due to the OCCURS clause, must not contain VALUE clauses. See example 1, below.

b. When the elements of the table do not require separate handling, the VALUE of the entire table may be given in the entry defining the entire table. The lower-level entries will show the hierarchical structure of the table; lower-level entries must not contain VALUE clauses. See example 2, below.

Example 1:

```
01  W-S-TOTS.
    03 FILLER SIZE 24 VALUE IS ZEROS.
    03 CARDIMAGE-VALUES SIZE 80.
01  R-TOTS REDEFINES W-S-TOTS.
    03 TOT PIC 9(4) OCCURS 26 TIMES.
```

Example 2:

```
03  FACTORS SIZE 25 VALUE "100031001110032100961018 4".
    05 Y-FACTOR PIC 9(5) OCCURS 5 TIMES.
```

Burroughs COBOL CODING FORM

PROGRAM: CODING OF MULTI-DIMENSIONED TABLE | REQUESTED BY: | PAGE 1 OF 1

PROGRAMMER: WENDY | DATE 20 FEB | IDENT. 73 80

PAGE NO. 1 3	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		01	EN-VOLUME ØCCURS 50 TIMES.
	02			05 EN-INDEX PIC X(6).
	03			05 EN-PAGE ØCCURS 30 TIMES.
	04			10 EN-HEADING SIZE 12.
	05			10 EN-PARAGRAPH ØCCURS 15 TIMES.
	06			15 EN-SUBJECT SIZE 6.
	07			15 EN-LINE ØCCURS 8 TIMES.
	08			20 EN-WORD SIZ 6 ØCCURS 7 TIMES.
	09			
	10			
	11			
	12			
	13			
	14			
	15			
	16			
	17			
	18			
	19			
	20			

Figure 6-3. Coding of Multi-Dimensioned Table

SUBSCRIPTING

Subscripts can be used only when reference is made to an individual element within a table of like elements that have not been assigned individual data-names. (Refer to the OCCURS clause.)

The subscript can be represented by a numeric literal that is an integer, by a formula (whose result must be a positive value and will be truncated to an integer by the compiler), or by an identifier. The identifier must be a numeric elementary item that represents an integer.

The subscript may be signed and if signed must be positive. The lowest permissible subscript value is 1. This value points to the first element of the table. The next sequential elements of the table are pointed to by subscripts whose values are 2, 3, ... The highest permissible subscript value, in any particular case, is the maximum number of occurrences of the item as specified in the OCCURS clause. Violation of this rule will cause the object program to terminate with an INVALID INDEX message.

The subscript, or a set of subscripts, identifying the table element is enclosed in parentheses. The table element data-name appended with a subscript is called a subscripted data-name or an identifier. When more than one subscript appears within a pair of parentheses, the subscripts may be separated by commas and are written in the order of successively less inclusive dimensions of the data organization.

The general format for subscripting is:

$$\left\{\begin{matrix}\text{data-name}\\ \text{condition-name}\end{matrix}\right\} \quad \text{(subscript [,subscript] ...)}$$

For example, in figure 6-3, to reference the first volume, EN-VOLUME (1) is written. If data-name N contains the number of the volume desired, EN-VOLUME (N) is written. If the data item PAGE-NO contains the number of the page desired, then EN-HEADING (N, PAGE-NO) would reference the twelve-character page heading. To reference the heading on the following page, EN-HEADING (N, PAGE-NO + 1) could be used. The fifth EN-WORD of the second EN-LINE of the first EN-PARAGRAPH of the third EN-PAGE of the Nth EN-VOLUME would be referenced by EN-WORD (N, 3, 1, 2, 5).

Where qualification and subscripting are both required, the qualification is shown first, followed by the subscripting. For example, EN-PAGE OF EN-VOLUME (N, PAGE-NO). EN-PAGE (N, 3) OF EN-VOLUME is incorrect. For further restrictions, refer to the discussion of identifiers in this section.

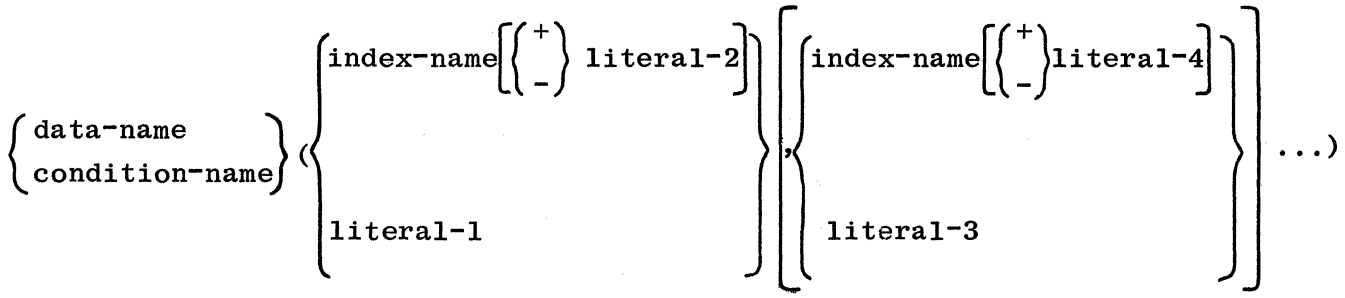
INDEXING

References can be made to individual elements within a table of like elements by specifying indexing for that reference. An index is assigned to that level of the table by using the INDEXED BY clause in the definition of a table. A name given in the INDEXED BY clause is known as an index-name and is used to refer to the assigned index. The value of an index corresponds to the occurrence number of an element in the associated table. An index must be initialized before it is used as a table reference. An index can be given an initial value by either a SET or a PERFORM statement.

Direct indexing is specified by using an index-name in the form of a subscript. Relative indexing is specified when an index-name is followed by the operator + or -, followed by an unsigned integer numeric literal all delimited by the balanced pair of separators left parenthesis and right parenthesis following the table element data-name. The occurrence number resulting from relative indexing is determined by incrementing (where the operator + is used) or decrementing (where the operator - is used), by the value of the literal, the occurrence number represented by the value of the index. When more than one index-name is required, they are written in the order of successively less inclusive dimensions of the data organization.

At the time of execution of a statement which refers to an indexed table element, the value contained in the index referenced by the index-name associated with the table element must neither correspond to a value less than one (1) nor to a value greater than the highest permissible occurrence number of an element of the associated table. This restriction also applies to the value resultant from relative indexing. Failure to observe this rule will cause the system to terminate the program if an attempt is made to SET index-name TO a value which is outside the bounds specified by the OCCURS clause. Using the UP BY or DOWN BY options of the SET statement does not invoke bounds checking for the SET but the program will still be terminated if an attempt is made to access beyond the end of the 01 record.

The general format for indexing is:



IDENTIFIER

An identifier is a term used to reflect that a data-name, if not unique in a program, must be followed by a syntactically correct combination of qualifiers, subscripts, or index-names necessary to ensure uniqueness.

The format for identifiers has two options which are as follows:

Option 1:

```
data-name-1 [{OF|IN} data-name-2] ... [{OF|IN} {file-name|report-name}]
                [( subscript-1 [, subscript-n]...)]
```

Option 2:

```
data-name-1 [{OF|IN} data-name-2] ... [{OF|IN} {file-name|report-name}]

[ ( {index-name-1 [ {+|-} literal-2]|literal-1} [, {index-name-2 [ {+|-} literal-4]|literal-3}]...) ]
```

Restrictions on qualification, subscripting, and indexing are as follows:

a. The commas as shown in both options are optional.
b. The data-name-2 must not itself be subscripted nor indexed.
c. Indexing is not permitted where subscripting is not permitted.
d. An index may be modified only by the SET, SEARCH, and PERFORM statements. Data items described by the USAGE IS INDEX clause permit storage of the values of index-names as data without conversion. Such data items are called index data items.
e. Where more than one occurrence number is required for a data name reference, it is illegal to use a data-name or formula subscript for one occurrence number and an index-name for another. However, literals and index-names may be mixed.

FILE DESCRIPTION ENTRIES

The function of the FILE SECTION is to describe the files. The format for this section contains three options which are as follows:

Option 1:

```
{FD|SD} file-name COPY library-name
                    [FROM sequence-number]
                    [[{THROUGH|THRU} sequence-number]]
                  [REPLACING word-1 BY text-1
                             [,word-2 BY text-2] ...].
```

Option 2:

```
FD file-name
[;RECORDING MODE IS {STANDARD|NON-STANDARD}]
[;BLOCK CONTAINS [integer-1 TO] integer-2 [CHARACTERS|RECORDS|WORDS]]
[;FILE CONTAINS integer-1 [BY integer-2] RECORDS]
[;RECORD CONTAINS [integer-3 TO] integer-4 [ASCII|COMP|COMPUTATIONAL|COMP-2|COMPUTATIONAL-2|DISPLAY|DISPLAY-1] [CHARACTERS|WORDS]]
[;LABEL {RECORD IS|RECORDS ARE} {STANDARD|OMITTED [WITH MULTIPLE AT END]|data-name-5 [,data-name-6]...}]
[;DATA {RECORD IS|RECORDS ARE} data-name-7 [,data-name-8]...]
[;LINAGE IS {integer-5|data-name-9} LINES]
[;SAVE-FACTOR IS {integer-6|data-name-10}]
[;{VA|VALUE|VALUES} OF {ID|IDENTIFICATION} IS {data-name-11|literal-1}]
  [;CODE-SET IS alphabet-name].
```

Option 3:

SD file-name

```
[; RECORD CONTAINS [integer-3 TO [CHARACTERS]]]
                   [WORDS     ]

[;DATA {RECORD IS } data-name-7 [,data-name-8]...].
       {RECORDS ARE}
```

For a discussion of the COPY function refer to section 8, THE COBOL LIBRARY.

A level indicator of FD or SD identifies the beginning of the file or sort-file description and precedes a unique file-name or sort-file-name.

The file-name is the highest level qualifier within an FD or SD entry and its associated record descriptions.

MCP control statements which reference a file of a program must specify the file-name which follows the FD. The MCP limits the number of characters in a file-name to 17. If more than 17 characters are specified in an MCP control statement, only the first 17 (leftmost 17) characters are used by the MCP. (Thus, if file-name is to be used in MCP control statements, the leftmost 17 characters must be unique.)

All semicolons are optional in the file description, but each complete description entry must be terminated by a period.

The clauses which follow the name of the file are optional in many cases, and their order of appearance is immaterial. Option 1 is used when the COBOL library contains the file description entry; otherwise, option 2 or option 3 is used.

All data-names used in file description entries may be qualified; however, they may not be subscripted or indexed.

All files selected in the FILE-CONTROL paragraph of the ENVIRONMENT DIVISION must have a file description entry.

BLOCK

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

The format for the BLOCK clause is as follows:

$$\underline{\text{BLOCK}}\ \text{CONTAINS}\ [\text{integer-1}\ \underline{\text{TO}}]\ \text{integer-2}\ \left\{\begin{array}{l}\text{CHARACTERS}\\ \underline{\text{RECORDS}}\\ \underline{\text{WORDS}}\end{array}\right\}$$

The BLOCK clause is optional and should not be used if the logical records of the file are unblocked; i.e., each physical block contains one logical record.

The clause may not be used with a SORT-FILE description or when DIRECT is specified in the SELECT clause for the file.

For object program efficiency, the use of blocked records is recommended for files assigned to disk or tape.

Fixed-size records (fixed size blocks) are handled more efficiently than variable-size blocks (variable-size records).

If only integer-2 is specified, a fixed-size block of size integer-2 is created or expected. The block size may be stated in either CHARACTERS, WORDS, or RECORDS.

If integer-1 is specified a variable FILETYPE file will be created or expected with integer-2 representing the maximum physical record size. The operating system will write or read as many variable-length logical records as possible to or from the maximum physical record size. A file is blocked only when maximum blocksize is larger than maximum record size. It will be created as an unblocked file when maximum blocksize is less than or equal to maximum record size. The location of a field that is to contain the size of the record written at object time is denoted in one of the following ways:

1. If a SIZE DEPENDING ON data-name is indicated on the 01 level-number in the record description (or PICTURE DEPENDING option if the 01 level is elementary), then at each WRITE, the operating system will assume that the data-name contains an integer representing the number of characters to be written. If data-name specifies the data item which occupies the first four INTMODE characters of the record, the file attribute FILETYPE will be 1. If data-name is not a four character item and/or data-name does not reference the first four characters of the record (but data name is defined within the record description), the action taken by the operating system is equivalent to the file

attribute FILETYPE = 4. If data-name is defined external to the record description, the action is equivalent to FILETYPE = 3.

2. If no SIZE DEPENDING or PICTURE DEPENDING is indicated in the 01 level and the BLOCK clause specifies integer-1 and integer-2, it is assumed that the first four characters of the record will contain the number of characters to be read or written (equivalent to the file attribute FILETYPE = 1).

Where there are multiple record descriptions for variable-length records, each must have the DEPENDING clause specifying the same data-item, or each must have no DEPENDING clause.

If the DEPENDING clause specifies a name inside the record, better output action will result.

The BLOCK clause of the FD and the SIZE clause of the 01 level Record Description are the only bases for determining the FILETYPE attribute.

When a file is assigned to disk, the user should be aware that the physical disk segment size is 30 words and that all physical READs and WRITEs will be in multiples of this size; therefore, it is preferred that the block size used be a multiple of 30 words.

The minimum block size for TAPE files is six words. When records are described which are less than six words, the final block will be padded with hex zeros if the block is less than six words. When such a block is input to a program, it may result in extra records containing hex zeros being made available. Thus, in a fixed length blocked file, such padded records may show upon input when MAXRECSIZE is three words or less.

Data Records

The DATA RECORDS clause is optional and serves only as documentation for the names of data records and their associated file. The format for the DATA RECORDS clause is as follows:

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

No syntax error will occur when a record declared for the file is not listed in the DATA RECORDS clause.

FILE

The function of the FILE CONTAINS clause is to specify either the total number of records in a file, the areas in a file, or the area size of each area in a file. The format for the FILE CONTAINS clause is as follows:

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

If integer-2 is not specified, integer-1 refers to the total number of records in a file. If integer-2 is specified, integer-1 refers to the number of areas, and integer-2 refers to the area size of each area in a file. The FILE CONTAINS clause provides compatibility with B 3700 COBOL and is available only while the B 2500 system dollar option is set. Refer to Appendix C for a description of the B 2500 implementations.

LABEL

The LABEL clause is used to specify the presence or absence of label information.

The format for this clause is:

```
LABEL { RECORD IS  }  { STANDARD                                  }
      { RECORDS ARE }  { OMITTED  [WITH MULTIPLE AT END]          }
                       { data-name-1 [,data-name-2]...            }
```

If the LABEL clause is not used, STANDARD is assumed. OMITTED must be used if an input file does not have standard labels or if labels are not desired on output files. If the WITH MULTIPLE AT END phrase is specified, a CLOSE filename with NO REWIND followed by an OPEN will bypass the tape mark.

STANDARD or data-name-1 should be used if the user wishes to take advantage of the automatic file allocation and handling procedures in the operating system. (Disk devices maintain a directory instead of a system of labels.) The format of labels is dependent upon the device containing the file. (See B 6700/B 7700 System Software Handbook, Form No. 5000722, for label formats.) The file handling procedures for tape recognize either the B 5500/B 5700 standard label, the B 2500/B 4700 standard label, the IBM 360/IBM 370 (USASI) standard label, or the B 7000/B 6000 standard label on input files, and produce the B 7000/B 6000 standard label on output files. The B 7000/B 6000 standard label for tape is in a format compatible with the proposed USASI standard label for information exchange.

The format data-name-1 [,data-name-2] should be specified if user header and trailer records are to be included in the B 7000/B 6000 standard tape label. While these labels may be specified for any device, they will not be written or read for files other than magnetic tape. The contents of user header and trailer labels are accessed by user-supplied label USE routines. These USE routines are performed by the operating system's file handling routines on file open, file close, and at volume (reel) switching time.

If user labels are specified, each must be exactly 80 characters in length and must be the same USAGE (DISPLAY or DISPLAY-1) as the first record description for the file. The first four characters of the label are reserved for the operating system. Figure 6-4 is an example of user label record coding.

```
       LABEL RECORDS ARE XYZ, RST
       VALUE OF ID IS "STN0235";
       DATA RECORDS ARE DATA1.
01     XYZ        SIZE 80.
       02  COCODE          PIC 9(4).
       02  DIVCODE         PIC 9(8).
       02  SECTCODE        PIC 9(3).
       02  REQ-BY          PIC X(65).
01     DATA1.
       02  ...
       02  ...
01     RST.
       02  ...
       02  ...
```

Figure 6-4. Label Coding

The user can specify the creation of single file volumes or multi-file volumes. In addition, the operating system will do volume switching for either of the above cases when the data being written exceeds the capacity of a volume. It will also do automatic volume switching on input when required. The tape format is shown as follows: (Note that * denotes a tape mark and ** denotes end-of-reel.)

a. Single-File, Single Volume (Single Reel File)

```
VOL1   HDR1   HDR2    *    DATA * EOF1    EOF2   **
```

b. Multi-Volume File (Multiple Reel File)

```
VOL1   HDR1   HDR2    *    FIRST VOLUME DATA * EOV1   **
VOL1   HDR1   HDR2    *    LAST VOLUME DATA * EOF1     EOF2  **
```

c. Multi-File Volume (Multiple File Reel)

```
VOL1   HDR1   HDR2    *   FILE 1 * EOF1   EOF2  *
       HDR1   HDR2    *   FILE 2 * EOF1   EOF2  **
```

d. Multi-File, Multi-Volume (Multiple File-Multiple Reel)

```
VOL1   HDR1   HDR2    *   FILE 1 * EOF1   EOF2  *
       HDR1   HDR2    *   FIRST PART FILE 2 * EOV1  **
VOL1   HDR1   HDR2    *   PART OF FILE 2 * EOV1  **
VOL1   HDR1   HDR2    *   REMAINDER FILE 2 * EOF1   EOF2  *
       HDR1   HDR2    *   FILE 3 * EOF1   EOF2  **
```

User header labels may appear immediately after HDR2, and user's trailer labels may appear after either EOF2 or EOV1.

To create or read multi-file volumes, the user must specify the same volume name for all the files in the set. Only one file in the set can be opened at a time. To create a multi-file volume, the user must CLOSE NO REWIND the current file in the set and use OPEN OUTPUT NO REWIND for the next file in the set. To handle input, the operating system will give back to the object code an END-OF-FILE condition when an EOF label is encountered. The user then must CLOSE NO REWIND on the current file and OPEN INPUT NO REWIND on the next (or some other) file in the set.

The EOV label, when encountered on input, is the sentinel by which the operating system can detect when volume switching is required. This is done by locating or requesting the operator to load a volume which has the same volume name as the current volume and has a file section number (in HDR1) one greater than the current volume.

After a tape mark has been read or written, normal action for unlabeled tape files is a "reel switch," i.e., the tape mark is taken as an END OF REEL condition. The LABEL clause specifying WITH MULTIPLE AT END allows the file to be CLOSED WITH NO REWIND, re-OPENed, and reading or writing to be continued.

LINAGE

The LINAGE clause specifies logically the number of lines to be written on a printer page. It is ignored if the file is assigned to a device other than a printer. The format for this clause is:

$$\underline{\text{LINAGE}} \text{ IS } \left\{\begin{matrix}\text{integer} \\ \text{data-name}\end{matrix}\right\} \text{ LINES}$$

When data-name is used, the data description must be that of a numeric elementary item without any positions to the right of the assumed decimal point.

The LINAGE clause provides a means of specifying the depth of a printed page; the printed page may or may not be equal to the physical perforated continuous form often associated with the page length. LINAGE is not permitted in the same file description entry with the REPORT clause.

The value of integer, as specified in the LINAGE clause, will be used at the time the file is opened by the OPEN statement to specify the number of lines (written and/or spaced) on a printed page and to set the PAGESIZE attribute. Integer must not be less than one.

The value of data-name, as specified in the LINAGE clause, will be used at file OPEN and at each end-of-page to specify the number of lines (written and/or spaced) for the next printed page.

A LINAGE-COUNTER is generated by the presence of the LINAGE clause. LINAGE-COUNTER is synonomous with the LINENUM attribute. The rules governing the LINAGE-COUNTER are as follows:

a. One LINAGE-COUNTER is supplied for each file described in the FILE SECTION whose FD entry includes the LINAGE clause.
b. A LINAGE-COUNTER may be used as a numeric source operand by PROCEDURE DIVISION statements. If more than one file has a LINAGE clause, then all references to LINAGE-COUNTER must be qualified by the file name.
c. The LINAGE-COUNTER may not be used as a receiving operand by PROCEDURE DIVISION statements.
d. LINAGE-COUNTER is automatically incremented by one each time a WRITE statement is executed for the associated file, with the following exceptions:
    1. When the ADVANCING option of the WRITE statement is used with the identifier-2 LINES option, the increment is the value of identifier-2.
    2. When the ADVANCING option of the WRITE statement is used with integer LINES option, the increment is the value of integer.

3. LINAGE-COUNTER is automatically reset to one when the ADVANCING PAGE phrase of the WRITE statement is specified.
4. LINAGE-COUNTER is reset to one when the file is positioned to a new page.

e. LINAGE-COUNTER is automatically set to one initially by the OPEN statement.
f. The value of the LINAGE-COUNTER at any given time represents the last line number printed or spaced on a logically printed page.
g. If the WRITE ADVANCING CHANNEL option is used, the contents of LINAGE-COUNTER will be unpredictable.

The "AT END-OF-PAGE" clause can be used with WRITE statements for any file assigned to a printer, regardless of whether or not a LINAGE clause is declared with the file. The END-OF-PAGE condition will occur as the last line is written, if LINAGE is specified or the attribute PAGESIZE is set to some value other than zero.

RECORD

The function of the RECORD clause is to indicate, in the absence of Record Descriptions, the size of data records for the file.

The format for the RECORD clause is as follows:

```
RECORD CONTAINS [integer-1 TO] integer-2 [ASCII|COMP|COMPUTATIONAL|COMP-2|COMPUTATIONAL-2|DISPLAY|DISPLAY-1] [CHARACTERS|WORDS]
```

Direct files may use the RECORD clause or a **Record Description** to specify the internal mode of the file. Only direct files can specify USAGE in the RECORD clause; that is, non-direct files cannot specify USAGE.

For non-direct files, the size of each data record is completely defined within the Record Description entry; therefore, this clause need not be specified.

For direct files, the size of data records must be specified in the RECORD clause and/or in the record descriptions which may follow the FD.

When record descriptions are present, the maximum record size is obtained from the length of the largest data record, in terms of the internal character size of the file; therefore, the RECORD clause is ignored. The RECORD clause is not a determining factor with regards to the MAXRECSIZE AND FILETYPE attributes.

When the RECORD clause is present, the following rules apply:

a. The presence of a RECORD CONTAINS [integer-1 TO] integer-2 ..., clause will not, by itself, cause the file to be declared as variable length (that is, a non-zero FILETYPE attribute).

b. If the maximum record size declared by the RECORD CONTAINS clause is different than the length of the largest record description, the compiler will emit a warning message declaring that the size specified by the RECORD CONTAINS clause is ignored.

c. The maximum record size cannot exceed the limits shown in the BLOCK CONTAINS clause.

d. If the file is assigned to DIRECT hardware-name in its SELECT clause, the maximum record size is determined by the RECORD CONTAINS clause in the absence of record descriptions and the [integer-1 TO] option is not permitted.

e. The usage of the first record described determines the internal mode (INTMODE) of the file.

f. When multiple record descriptions are used, the first record described determines the internal mode of the file. For non-direct files, however, the first record description for a file cannot have a USAGE of COMP-2 (4-bit characters), since the operating system will not handle 4-bit character INTMODE.

g. Files having a BLOCK CONTAINS integer-1 TO integer-2 clause will be considered as having a variable length FILETYPE.

h. If the 01-level record descriptions for the file do not specify a size clause indicating a variable length item, then the 01-level data items will be considered as fixed length (i.e., the value of the first four characters in the record will only be used by logical I/O to determine the length of a written record not in determining the length of the 01-level item).

The value returned by the file attribute MAXRECSIZE may be the size of the record in words if records are some multiple of word size.

The minimum block size for files assigned to TAPE is six words. When a record of a TAPE file is less than six words, the last block will be padded with hex zeros if it contains less than six words. When such a block is input to a program, it may result in extra records containing hex zeros being made available. The MCP will discard any record of a fixed length blocked file whose size is less than MAXRECSIZE. Thus, in a fixed length blocked file, such padded records may only show up on input when MAXRECSIZE is three words or less.

Figure 6-6 (Variable-Length Blocked Records Coding) demonstrates various methods for describing variable length records. The file DCG could contain as many as 13 records in a block. File DEC could contain only a single record in a block if that record (or the next record) were the maximum size record (i.e., 5288 characters). File WSG is unblocked and its records, instead of being variable length, are referred to as fixed length records of differing sizes (i.e., either 84, or 168 or 252 characters).

(This page deleted)

RECORDING MODE

This clause is used to specify the RECORDING MODE for peripheral devices where a choice can be made.

The format for the RECORDING MODE clause is:

RECORDING MODE IS { STANDARD / NON-STANDARD }

STANDARD recording mode is assumed if this clause is not present. The RECORDING MODE's for the peripheral devices on the B 7000/B 6000 are:

DEVICE	STANDARD	NON-STANDARD
CARD READER	ALPHA	--
PUNCH	ALPHA	--
PRINTER	ALPHA	--
7-TRACK TAPE	BINARY (ODD PARITY)	ALPHA (EVEN PARITY)
9-TRACK TAPE	EBCDIC (ODD PARITY)	--
PAPER-TAPE	ALPHA (ODD PARITY)	BINARY (ODD PARITY)
DISK	BINARY	--

When ALPHA recording mode is used, there will be an automatic translation from BCL to EBCDIC on input, EBCDIC to BCL on output. For example, if an input file is in BCL but the record description is EBCDIC, automatic BCL-to-EBCDIC translation takes place.

Binary files are read or written as 48-bit words, with no possibility of translation.

SAVE-FACTOR

The SAVE-FACTOR clause specifies the number of days that an output file is to be saved.

The format for this clause is as follows:

SAVE-FACTOR IS { integer / data-name }

SAVE-FACTOR need not be specified for input files. For output tape files, the integer is the number of days the file is to be saved before the tape can be reused by the B 7000/B 6000 system. This integer is used to generate the expiration date on tape labels; it must be unsigned and cannot exceed three digits.

For tape, if the expiration data is previous to the current data and the tape has a Write Ring, then the tape is marked as scratch.

Data-name may be used to assign the integer value of the save-factor at object time. This is done by executing an explicit OPEN statement for the file.

VALUE

VALUE

The VALUE clause specifies the external name (TITLE) of a labeled file in the label records associated with a file.

The format for the VALUE clause is as follows:

$$\left\{\begin{matrix}\underline{VA}\\ \underline{VALUE}\\ \underline{VALUES}\end{matrix}\right\} \text{ OF } \left\{\begin{matrix}\underline{ID}\\ \underline{IDENTIFICATION}\end{matrix}\right\} \text{ IS } \left\{\begin{matrix}\text{data-name}\\ \text{literal [/ literal] ...}\end{matrix}\right\}$$

The VALUE OF ID IS literal option is used when the actual identification of the file is known in advance. The file-title literal can be in any of the following forms:

OPTION A [VOLUME-ID /] FILE-ID

OPTION B VOLUME-ID / FILE-ID

OPTION C DIRECTORY-ID [/ DIRECTORY-ID] ... / FILE-ID

where DIRECTORY-ID, VOLUME-ID and FILE-ID are non-numeric literals, one to seventeen characters in length and should not contain any special characters or spaces so that operational considerations of the MCP will not become a problem.

Examples:

```
VALUE OF ID "ABC" / "XYZ"
VALUE OF IDENTIFICATION "INPUTCARDDATA"
```

An external name may contain up to 14 identifiers.

Option A is for single-volume or multi-volume files (VOLUME-ID is filled with zeros).

Option B is for multi-file volumes or multi-file multi-volumes, where all files with the same volume-ID are expected/created on the same reel.

Option C is for files that are controlled via the disk directory or for tape files. When used for tape files, the first DIRECTORY-ID is used as the VOLUME-ID and all other DIRECTORY-ID's are ignored. For details, see B 7000/B 6000 INPUT/OUTPUT SUBSYSTEM REFERENCE MANUAL, Form No. 5001779.

The VALUE OF ID data-name option is used when the actual identification of the file is set by PROCEDURE DIVISION statements. The proper identification must be moved into the data-name prior to opening of the file. Data-name must be DISPLAY and its contents must be one to 14 names of 17 or less characters each, separated by slashes (/) with a period (.) following the last name. An explicit OPEN statement must be executed to cause the external name (TITLE) to be obtained from data-name. When VALUE-OF-ID is not specified, file-name will be utilized as both external name (TITLE) and internal name (INTNAME).

Example:

```
MOVE "ABC/DEC." TO DATA-NAME.
```

CODE-SET

The CODE-SET clause is an ANSI 74 extension. CODE-SET is allowed in the FD entry of any non-direct, non-indexed file.

The format for the CODE-SET clause is as follows:

[; CODE-SET IS alphabet-name]

When the CODE-SET clause is specified, all data items in the record descriptions for the file must be described as usage DISPLAY. The EXTMODE attribute of the file is initialized in the file description with the specified value and the TRANSLATE attribute is initialized to VALUE (FULLTRANS).

The alphabet-name clause referenced by the CODE-SET clause must not be that of Format 2.

If the CODE-SET clause is not specified, no initial values are given the EXTMODE and TRANSLATE attributes in the file description.

CODE-SET is available when the **ANSI74 system dollar option is set.** Refer to Appendix B for a description of the ANSI 74 implementations.

Figure 6-5 illustrates the manner in which the FILE SECTION is coded.

Burroughs COBOL CODING FORM

PROGRAM: FILE SECTION Coding — REQUESTED BY: — PAGE 1 OF 1

PROGRAMMER: DEE — DATE: — IDENT. 73 80

PAGE NO.	LINE NO.	A	B
	01	FILE SECTION.	
	02	FD	PERSONNEL-MASTER
	03		BLOCK 3 RECORDS RECORDING MODE NON-STANDARD
	04		VA OF ID "PERSONNEL" / "MASTER" SAVE-FACTOR 10.
	05	01	EMPL-RECORD SIZE 84.
	06		05 EMPL-NUMBER PIC 9(8).
	07		05 DEPT PIC 999.
	08		05 FILLER SIZE 9.
	09		05 JOB-CODE PIC XXXX.
	10		05 FILLER PIC X(60).
	11		
	12	SD	EXTRACT-FILE.
	13	01	SORT-REC.
	14		05 S-JOB-CODE SIZE 4.
	15		05 S-EMPL-NUMBER PIC 9(8).
	16		05 FILLER SIZE 3.
	17		
	18	FD	DEPT-REPORT VA OF ID "PERSONNEL" / "REPORT".
	19	01	HEADING-LINES SIZE 132.
	20	01	BODY-LINES SIZE 132.
			10 JOB-CODE PIC X(4) B(4).
			10 EMPL-COUNT PIC Z(5)9.
			10 FILLER SIZE 118.
		FD	REPORTER VA OF ID "REPT-1" REPORTS IS RD-1.

Figure 6-5. FILE SECTION Coding

Figure 6-6 is an example of coding for variable-length blocked records

Burroughs COBOL CODING FORM

PROGRAM: VARIABLE-LENGTH BLOCKED RECORDS CODING | REQUESTED BY | PAGE 1 OF 1
PROGRAMMER: PAT | DATE | IDENT. 73 80

PAGE NO.	LINE NO.	A	B
	01	FILE-CONTROL.	SELECT DEC ASSIGN TO TAPE.
	02		SELECT DCG ASSIGN TO TAPE.
	03		SELECT WSG ASSIGN TO TAPE.
	04		
	05		
	06	FD DCG	BLOCK 3620 TO 4000 CHARACTERS VA OF ID "KMC".
	07	01 KIM	SIZE 300 TO 400 DEPENDING ON KIM-REC-SIZE.
	08	05	KIM-REC-SIZE PIC 9(4).
	09	05	TLR-OF-KIM PIC 9(3).
	10		
	11		
	12	FD DEC	BLOCK 2088 TO 7224 CHARACTERS VA OF ID "WSG".
	13	01 KEY	SIZE 2088 TO 5288 DEPENDING ON KEY-REC-SIZE.
	14	05	KEY-REC-SIZE PIC 9(4).
	15	05	REST-OF-KEY SIZE 5284.
	16	FD WSG	VALUE OF ID "CJC"/"BLC".
	17	01 SHORT-REC	SIZE 84.
	18	01 MEDIUM-REC	SIZE 168.
	19	01 LONG-REC	SIZE 252.
	20		

Figure 6-6. Variable-Length Blocked Records Coding

RECORD DESCRIPTION

The RECORD description portion follows the FILE description entries and, as will be shown, serves to completely identify each of the records in the file.

Each FD or SD entry must be followed by at least one 01 level record description, except for direct files which may use the RECORD clause to specify the internal mode and record size of the file. The FD for a direct file does not require the presence of a record description.

The format for the Record Description entry consists of the following four options:

Option 1:

```
01 data-name-1 ; COPY library-name

    [ FROM seq. no.][{ THRU     } seq. no.]
                    { THROUGH }

    [REPLACING  word-1 BY text-1
        [, word-2 BY text-2] ...].
```

Option 2:

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

[;SEGMENT]

[;{ SIZE } IS [integer-1 TO] integer-2 CHARACTERS [DEPENDING ON data-name-3]]
  { SZ   }

[; { PICTURE } IS character-string [DEPENDING ON data-name-4]]
   { PIC     }

[;GLOBAL]
[;LOCAL]
[;OWN]

[;[USAGE IS] { COMP                 { ASCII                 }
             { COMPUTATIONAL        { DISPLAY               }
             { COMP-1               { DISPLAY-1             }
             { COMPUTATIONAL-1      { INDEX                 }
             { COMP-2               { EVENT                 }
             { COMPUTATIONAL-2      { LOCK                  }
             { COMP-4               { CONTROL-POINT         }
             { COMPUTATIONAL-4      { CP                    }
             { COMP-5
             { COMPUTATIONAL-5
             { INDEX FILE [CONTAINS file-name-1 [,file-name-2]...] ]
```

```
[;{OC      }[integer-3 TO] integer-4 TIMES [DEPENDING ON data-name-4]
  {OCCURS  }
    [{ASCENDING } KEY IS data-name-5 [,data-name-6]...]...
     {DESCENDING}
    [INDEXED BY index-name-1 [,index-name-2]...]]

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

[;{JUSTIFIED} RIGHT]
  {JUST     }

[;RANGE IS literal-1 {THROUGH} literal-2]
                    {THRU   }

[;RECEIVED BY {REFERENCE}]
              {REF      }
              {CONTENT  }

[;BLANK WHEN ZERO]

[;{VA    }  [IS ]  literal-1]
  {VALUE }  [ARE]
  {VALUES}

[;RECORD AREA]

[;WITH {LOWER-BOUNDS}].
       {LOWER-BOUND }
```

Option 3:

```
66 data-name-1 ; RENAMES data-name-2 {THROUGH} data-name-3 .
                                     {THRU   }
```

Option 4:

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

[ ,literal-3 [ { THROUGH } literal-4 ] ]... .
               { THRU    }
```

All semicolons and commas are optional in the **record** description, but the entry must be terminated by a period.

Level-number in option 2 may be any integer from 01 thru 49, or 77. The clauses may be written in any order, with two exceptions: the data-name-1 or FILLER clause must immediately follow the level-number; the REDEFINES clause, when used, must immediately follow the data-name-1 clause.

Either the SIZE or PICTURE clause must be specified for every elementary item except an index data item, EVENT item, LOCK item, COMP-4, COMP-5, or CP item, in which case the use of these clauses is prohibited.

The clauses SYNCHRONIZED, PICTURE, JUSTIFIED, RANGE, and BLANK WHEN ZERO must not be specified except at the elementary item level.

Option 4 cannot be used in the CONSTANT SECTION.

Option 4 is used for each condition-name. Each condition-name requires a separate entry with level-number 88. Option 4 contains the name of the condition and the value, values, or range of values associated with the condition-name. The condition-name entries for a particular conditional-variable must follow the entry describing the item with which the condition-name is associated. A condition-name can be associated with any item containing a level-number except the following:

a. Another condition-name.
b. A level 66 item.
c. A group containing items with descriptions including JUSTIFIED, SYNCHRONIZED, or USAGE (other than USAGE IS DISPLAY or DISPLAY-1.)
d. An index data-item.

Condition-names can be declared associated with group data items. Such condition-names are alphanumeric in class and cannot use numeric literals in their value clauses.

The record description clauses and level numbers are discussed in alphabetical order on the following pages.

Multiple level 01 entries subordinate to a File Description entry represent implicit redefinitions of the first record defined for the file.

BLANK WHEN ZERO

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

The format for the BLANK WHEN ZERO clause is as follows:

BLANK WHEN ZERO

This clause cannot be used for variable-length items.

When this clause is used, the data-item will contain spaces if the value of the item was all zeroes.

This clause can only be used with items whose PICTURE is specified as numeric edited or numeric. The category of the item is numeric edited.

Condition-Name

Condition-name is a special name which the user assigns to one or more values which a particular data field may assume. These values may then be referred to by the condition-name. The data field itself is called a conditional variable and must immediately precede the condition-name entries.

The format for condition-name is:

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

   [ ,literal-3 [ { THROUGH } literal-4 ] ]... .
                  { THRU    }
```

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

As an example of the use of condition-names, suppose one is working with data relating to school children where each grade is to be given a value; that is, the first grade value is 1, second grade 2, and so on thru the 12th grade, which has a value of 12. In addition, the grouping of grades 1 thru 6 will be called GRADE-SCHOOL, grades 7 thru 9 JUNIOR-HIGH, and grades 10 thru 12 HIGH-SCHOOL. Any other numerical values for the condition-names GRADE-SCHOOL, JUNIOR-HIGH, or HIGH-SCHOOL would be in error. Figure 6-7 is an example of how this may be shown in COBOL.

In a COBOL procedure, the programmer may now test for a condition-name. For IF SECOND-GRADE, the compiler will generate the coding necessary to test the item GRADE with the value 02. The test, IF HIGH-SCHOOL, will cause the testing of GRADE with the values 10, 11, and 12. The test, IF GRADE-SCHOOL, is equivalent to writing IF GRADE IS GREATER THAN ZERO AND LESS THAN 07.

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

a. In a condition-name entry, the VALUE clause is required.
b. The VALUE clause and the condition-name itself are the only two clauses permitted in the entry.

c. The characteristics of a condition-name are implicitly those of its conditional variable.

d. Whenever the THRU entry is used, literal-1 must be less than literal-2, literal-3 less than literal-4, etc.

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

f. Condition-names may be declared for either group or elementary items.

g. Condition-names, when specifying figurative constants, must specify figurative constants which are consistent with the class of the conditional variable.

NOTE: Numeric display data items are allowed to be compared alphanumerically in a relation condition with a "non-numeric" figurative constant, although such figurative constants are not allowed in a condition-name declaration associated with a numeric conditional variable. The rules for acceptability of a figurative constant in a condition-name declaration are the same as those for the initializing VALUE clause (numeric items are allowed: ZERO, ZEROS, ZEROES, LOWER-BOUND, LOWER-BOUNDS, UPPER-BOUND and UPPER-BOUNDS).

Burroughs COBOL CODING FORM

PROGRAM: CONDITION-NAME CODING | REQUESTED BY | PAGE 1 OF 1
PROGRAMMER: PAULINE | DATE | IDENT. 73 80

PAGE NO. 1 3 | LINE NO. 4 6 | 7 | A 8 11 | B 12 72 Z

```
01
02
03
04     05  GRADE                PIC 99.
05         88  FIRST-GRADE              VA 1.
06         88  SECOND-GRADE             VA 2.
07         88  THIRD-GRADE              VA 3.
08         88  FOURTH-GRADE             VA 4.
09         88  FIFTH-GRADE              VA 5.
10         88  6TH-GRADE                VA 6.
11         88  7TH-GRADE                VA 7.
12         88  8TH-GRADE                VA 8.
13         88  9TH-GRADE                VA 9.
14         88  10TH-GRADE               VA 10.
15         88  11TH-GRADE               VA 11.
16         88  12TH-GRADE               VA 12.
17         88  GRADE-SCHOOL             VA 1 THRU 6.
18         88  JR-HIGH                  VALUES 7, 8, 9.
19         88  HIGH-SCHOOL              VALUES 10 THRU 12.
20         88  GRADE-ERROR              VALUES ARE 13 THRU 99, 0.
```

4 8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72

Figure 6-7. Condition-Name Coding

Data-Name, FILLER

A data-name specifies the name of the data being described. The word FILLER specifies an item that cannot be referred to explicitly.

The format for the data-name clause is:

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

In the DATA DIVISION sections, a data-name or the key word FILLER must be the first word following the level-number.

Numeric FILLER items are allowed to declare more than 23 decimal places only if they do not have an initial VALUE clause and do not have any condition-names associated with them.

The structure of the data-name must conform to the rules for DATA-NAME found under NOUNS.

The key word FILLER may be used to name an unreferenced elementary item in a record, with the length of the named item being specified by a SIZE clause, etc. Under no circumstances can a FILLER item be referred to directly.

FILLER items are allowed to be group items only while the B2500 system dollar option is set. This feature provides system compatibility with B 3700 COBOL.

GLOBAL

COBOL procedures compiled at lexicographic level three or higher may use the untyped procedures, files, direct files, and certain variables in the outer, or D-2, block of the host program by declaring these to be GLOBAL.

Any 77 level stack operand or 01 level item that is declared in the WORKING-STORAGE SECTION of a host program and can be passed as a parameter can be declared global in a bound procedure by using the GLOBAL clause in its data description entry. GLOBAL declarations are matched by name and type to the global directory of the host. GLOBAL must not be specified in the host program or on a stack (COMP-1) array.

Examples:

```
77 GLASTATUS GLOBAL COMP-1 PIC 9(11).
77 BL-EVENT GLOBAL EVENT.
77 GL-SWFL INDEX FILE GLOBAL.
01 GL-RCD RECORD AREA GLOBAL OCCURS 10 SZ 180.
01 GL-EBCRAY GLOBAL.
   03 CMP-ITE COMP PIC 9(11) OCCURS 100 INDEXED BY I1.
```

A stack listing will show the addresses of global variables to have a lex level of two and a displacement of from two to N+1, where N is the number of global items declared. The displacements of references to these items are changed at bind time to match the actual displacements in the host.

Index-names generated for a GLOBAL array are not themselves GLOBAL items but are treated as if they had been described as OWN. Index-names for a LOCAL array will be treated as LOCAL variables.

If most or all of the variables declared in the WORKING-STORAGE SECTION are desired to be declared global, the $ option GLOBAL may be used. The $ option may be left set throughout the compilation, though it will only affect variables which are candidates for global declaration and are in the WORKING-STORAGE SECTION. The LOCAL clause or OWN clause may be used to override the $ option.

For example:

```
$ SET GLOBAL
        77 G1 COMP-1 PIC 9(11).
        77 G2 COMP-1 PIC 9(11).
        77 L1 LOCAL COMP-1 PIC 9(11).
        01 G3.
           03 FLD OCCURS 10 INDEXED BY I2.
```

In this example, G1, G2, and G3 will be declared GLOBAL, while L1 and I2 will be given an address local to the stack of the procedure.

JUSTIFIED

The JUSTIFIED clause specifies non-standard positioning of data within a receiving data item.

The format for the JUSTIFIED clause is as follows:

$$\left\{\frac{\text{JUSTIFIED}}{\text{JUST}}\right\} \text{ RIGHT}$$

The JUSTIFIED clause cannot be specified for a numeric-edited data item or for an item described as numeric. The JUSTIFIED clause cannot be specified for an item whose size is variable or for group items.

The following are the standard rules for positioning within an area:

a. Numeric data is aligned by decimal point (either implicit or explicit), with zeros filling any unused positions on either end, as required. In the absence of an explicit decimal point indication, the decimal point is assumed to be in the next position to the right of the units digit. Edited numeric data items are aligned by decimal point, with zero fill or truncation at either end as required within the receiving character positions of the data item, except where editing requirements cause replacement of the leading zeros.

b. Alphabetic or alphanumeric receiving data items are aligned at the leftmost character position in the data item, with space fill or truncation to the right.

When the receiving data items are described with the JUSTIFIED clause and it is larger than the sending item, the data is aligned at the rightmost character position in the data item, with leading space fill.

If JUSTIFIED RIGHT is specified for an alphabetic or alphanumeric item, data is placed into the area with space fill to the left.

If JUSTIFIED RIGHT is specified for an alphabetic or alphanumeric item and the receiving field is smaller than the sending field, truncation will occur from the left.

When standard justification is desired, the JUSTIFIED clause is not required. Justification is considered only when data is moved into an area.

LEVEL—NUMBER

Level-Number

The level-number indicates the hierarchy of data within a logical record and identifies entries for condition-names, non-contiguous constants, working storage items, and regrouping.

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

Data description entries subordinate to an FD or an SD entry may have level-numbers with the values 01 thru 49, 66, or 88. Data description entries subordinate to an RD entry may have level-numbers with the values 01 thru 49 only. Data description entries in the WORKING-STORAGE, CONSTANT, LINKAGE, and LOCAL-STORAGE sections may have level-numbers with the values 01 thru 49, 66,77 or 88.

Multiple level 01 entries subordinate to a particular level indicator in the file section represent implicit redefinition of the same area.

The use of level-number is as follows:

a. The level-number 01 identifies the first entry in each record description or report group.
b. Special level-numbers have been assigned to certain entries where there is no real concept of level:
    1. Level-number 66 is used with RENAMES entries. It must be used with option 3 of the data description skeleton.
    2. Level-number 88 is assigned to entries which define condition-names associated with a conditional variable and can only be used with option 4 of the data description skeleton. An entry with a level-number of 88 cannot be used in the CONSTANT SECTION.
    3. Level-number 77 is assigned to identify non-contiguous CONSTANT and WORKING-STORAGE items and can be used only in option 2 of the data description skeleton. (See the discussion on WORKING-STORAGE and CONSTANT SECTIONS.)

LOCAL

The LOCAL clause is discussed in the description of the GLOBAL clause.

LOWER-BOUNDS

The LOWER-BOUNDS clause permits COBOL programs to pass or receive array parameters with LOWER-BOUNDS.

The format for this clause is as follows:

$$\text{WITH} \left\{ \begin{array}{l} \underline{\text{LOWER-BOUNDS}} \\ \underline{\text{LOWER-BOUND}} \end{array} \right\}$$

This clause must be used when communicating with FORTRAN. The clause is used in the data description of an 01 item in WORKING-STORAGE (if LOWER-BOUNDS are received) or LOCAL-STORAGE (if LOWER-BOUNDS are to be passed).

All LOWER-BOUNDS passed out are zero. LOWER-BOUNDS received are not used in addressing the array. The purpose of this clause is to describe parameters compatible with FORTRAN and ALGOL. (The lowest bound of an array in ALGOL is zero. That item in a COBOL array is referenced as the first item.)

A LOWER-BOUND area must not be declared to be COMP-1.

OCCURS

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

The format for this clause has the following two options:

Option 1:

```
{OC     }
{OCCURS } integer-2 TIMES
    [ [ {ASCENDING } KEY IS data-name-2 [,data-name-3] ... ] ... ]
    [ {DESCENDING} ]
    [INDEXED BY index-name-1 [,index-name-2] ...]
```

Option 2:

```
{OC     }
{OCCURS } integer-1 TO integer-2 TIMES [DEPENDING ON data-name-1]
    [ [ {ASCENDING } KEY IS data-name-2 [,data-name-3] ... ] ... ]
    [ {DESCENDING} ]
    [ INDEXED BY index-name-1 [,index-name-2] ... ]
```

Integer-1 and integer-2 must be positive integers. If both are used, the value of integer-1 must be less than integer-2. The value of integer-1 may be zero, but integer-2 cannot be zero.

The data description of data-name-1 must describe a positive integer.

Data-name-2 must either be the name of the entry containing the OCCURS clause or the name of an entry subordinate to the entry containing the OCCURS clause.

Data-name-3, etc., must be the name of an entry subordinate to the group item which is the subject of this entry.

Data-name-1, data-name-2, and data-name-3 may be qualified.

OCCURS

The OCCURS clause cannot be specified in a data description that:

a. Has a 66, 77, or 88 level-number.

b. Describes an item whose size is variable. The size of an item is variable if its data description, or any item subordinate to it, is described by option 2 of the SIZE clause or has an L in the PICTURE clause.

The OCCURS clause is used in defining tables and other homogeneous sets of repeated data. Whenever the OCCURS clause is used, the data-name which is the subject of this entry must be either subscripted or indexed whenever it is referred to in a statement other than SEARCH. Further, if the data-name associated with the OCCURS clause is the name of a group item, then all data names belonging to the group must be subscripted or indexed whenever they are used in a statement other than SEARCH or as the object of a REDEFINES claus

Except for the OCCURS clause itself, all data description clauses associated with an item whose description includes an OCCURS clause apply to each occurrence of the item described.

A level 01 data item with an OCCURS clause may not have a REDEFINES clause.

The OCCURS clause can be used in the WORKING-STORAGE, LINKAGE and LOCAL-STORAGE sections on 01 level entries. This creates a two-dimensional array that is handled as such by the hardware. The first subscript can only be a numeric literal, non-subscripted identifier, or an index-name +/-literal. This type of array may not be redefined nor may it be COMP-1.

In option 1, the value of integer-2 represents the exact number of occurrences of items within the table.

In option 2, the value of integer-1 represents the minimum number of occurrences, and integer-2 represents the maximum number of occurrences. This does not imply that the length of the table is variable but that the number of occurrences is variable.

Any unused character positions resulting from the DEPENDING option will appear in the external media.

The DEPENDING option is only required when the end of the occurrences of data items cannot otherwise be determined. The value of data-name-1 is the count of the number of occurrences of items, and its value must not exceed integer-2. Reducing the value of data-name-1 makes inaccessible the contents of the indicated data items whose subscripts exceed the new value of data-name-1.

The implicit qualification of a "DEPENDING ON" variable in an OCCURS DEPENDING clause by the 01-level record name is allowed only while the B2500 system dollar option is set.

When the ANSI74 system dollar option is set, the specification of an OCCURS DEPENDING clause causes all group items subsuming the OCCURS clause to be considered as variable length items whose length depends indirectly on the value of the "DEPENDING ON" variable. In this case, any unused character positions resulting from the DEPENDING option will not appear in the external media. Refer to Appendix B for a description of the ANSI 74 implementations.

When the ANSI74 option is reset, group items subsuming the OCCURS DEPENDING clause are not considered as variable length items, and references to them will reference the entire table.

In the following example, after the MOVE statement is executed, F1 would contain AB; F2(1) and F2(2) would contain CD and EF, respectively. The area physically occupied by F2(3) would contain GH; however, since the value of CNT is 2, F2(3) logically does not exist, and an attempt to reference it will cause an error termination.

Example:

```
77 CNT PIC 9 VALUE 2.
01 DATA-T.
        03 F1 PIC XX.
        03 F2 PIC XX OCCURS 1 TO 4 TIMES DEPENDING ON CNT.
        03 F3 PIC XX.
.
.
.
        MOVE "ABCDEFGH" TO DATA-T.
```

If data-name-1 in the DEPENDING option is an entry in the same record as the current data description entry, data-name-1 must not be the subject of, or be subordinate to, an entry whose description includes option 2 of an OCCURS clause. For example, the following is illegal:

```
01 W-S-TABLE.
        02 TAB SIZE 5 OCCURS 1 TO 5 TIMES DEPENDING ON DEP-NAME.
            03 DEP-NAME PIC 9.
            03 CODE PIC 9(4).
```

Record descriptions may be passed to or received from an ALGOL program. The ALGOL program must declare arrays which will yield a structure identical to that in the COBOL program:

```
01 JA OC 20 COMP.                    }
       05 KA OC 25.                  }
           10 LA OC 3.               }  ARRAY A[0:19, 0:74];
               15 MA PIC 99.         }
01 ABC OC 20 PIC X(24).              EBCDIC ARRAY B[0:19, 0:23];
01 XYZ SIZE 132.                     EBCDIC ARRAY C[0:131];
01 BCA PIC X(132) LOWER-BOUNDS.      EBCDIC ARRAY D[*];
```

The KEY IS option is used to indicate that the repeated data is arranged in ascending or descending order according to the values contained in data-name-2, data-name-3, and so on. The data-names are listed in descending order of their significance.

If data-name-2 is not the subject of this entry, then the following applies:

a. All of the items identified by the data-names in the KEY IS phrase must be within the group item which is the subject of the OCCURS entry.
b. None of the items identified by data-names in the KEY IS phrase can be described by an entry which either contains an OCCURS clause or is subordinate to an intervening entry which contains an OCCURS clause.

An INDEXED BY clause is required if the subject of this entry, or an item within it, is to be referred to by indexing. If indexing is to be used, each table dimension must contain an INDEXED BY clause. The index-names identified by the clause must not be defined elsewhere in the program and must be unique.

The maximum number of data items, not including FILLER items, which may be subordinate to a data item having an OCCURS clause is 511.

The following example illustrates a use of the OCCURS clause to provide nested descriptions. A reference to ITEM-6 requires the use of four levels of subscripting; e.g., ITEM-6 (2, 5, 4, 9). A reference to ITEM-3 requires two subscripts; e.g., ITEM-3 (I,J).

```
.          .          .
.          .          .
.          .          .
2 ITEM; OCCURS 2 TIMES;...
    3 ITEM-1;...
    3 ITEM-2; OCCURS 5 TIMES;...
        4 ITEM-3;...
        4 ITEM-4; OCCURS 5 TIMES;...
            5 ITEM-5;...
            5 ITEM-6; OCCURS 10 TIMES;...
            .          .          .
            .          .          .
            .          .          .
```

In the example above, there are 500 ITEM-6 quantities.

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 the record being described as containing 10 individual data items, it could be described in the following manner:

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

The above definition 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. (See the discussion on subscripts.)

OWN

COBOL procedures compiled at level three or higher may declare certain variables to be OWN. These variables will retain their value or state throughout repeated exit and re-entry of the procedure in which they are declared.

Any item that can be declared in the WORKING-STORAGE SECTION, except for DIRECT SWITCH FILES, can be made OWN either by using the OWN clause or by using the $ option OWN.

Example:

```
77 X PIC X(10) OWN.
77 Y REDEFINES X PIC 9(10).
01 A OWN.
    03 CMP-ITEM COMP PIC 9(11) OCCURS 100 INDEXED BY II.
```

OWN must not be specified for a stack (COMP-1) array.

A stack listing will show the address of own variables to have a lex level of two and a displacement of from M to M-N+1, where M is the maximum displacement that may be accessed by the procedure, and N is the total number of stack locations obtained for OWN items. The BINDER will obtain actual addresses for OWN items by extending the D2 stack of the host.

All related index-names and copy descriptors for OWN items are also OWN; redefinitions of OWN items are implicitly OWN and need not use the OWN clause.

Use of the $ option OWN throughout the compilation will cause all stack locations obtained in the WORKING-STORAGE SECTION, except for DIRECT files, and stack (COMP-1) arrays to be OWN, unless overridden temporarily by a GLOBAL or LOCAL clause on an individual item.

If the $ option OWN is set at the PROCEDURE DIVISION header, the special register TALLY will be made an OWN variable.

PICTURE

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

The general format for the PICTURE clause is as follows:

$$\left\{\frac{\text{PICTURE}}{\text{PIC}}\right\} \text{ IS character-string } [\underline{\text{DEPENDING}} \text{ ON data-name}]$$

A PICTURE clause can only be used at the elementary item level.

A character-string consists of certain allowable combinations of characters in the COBOL character set used as symbols. The allowable combinations determine the category of the elementary item.

The maximum number of symbols allowed in the character-string is 30. When an unsigned integer enclosed in parentheses immediately follows a symbol, the integer specifies the number of consecutive occurrences of that symbol. This may not be used for those symbols limited to one occurrence per picture.

The DEPENDING ON option is used to denote a variable length elementary item. Variable length items cannot be described in the REPORT SECTION. The data description of data-name must be such that it defines a positive integer. The value of data-name represents the number of characters in the item being described and may have a value of zero. Data-name may be qualified but cannot be a variable-length item. This clause may be specified only when the PICTURE character "L" is specified.

The DEPENDING ON clause is not permitted when the description of the item contains the JUSTIFIED clause.

Either a PICTURE or a SIZE clause must appear in every elementary item except those items whose USAGE is declared as COMP-4, COMP-5, INDEX, EVENT, CONTROL-POINT or LOCK. If both a PICTURE and a SIZE clause appear, the SIZE clause is ignored. The SIZE of an elementary item, where SIZE means the number of character positions occupied by the elementary item, is determined by the number of allowable symbols that represent character positions.

Categories of Data

There are five categories of data that can be described with a PICTURE clause: Alphabetic, numeric, alphanumeric, alphanumeric edited, and numeric edited. These categories are described as follows:

PICTURE

ALPHABETIC

To define an item as alphabetic, its PICTURE character-string can only contain the symbols A, B, and L, and its contents, when represented externally, must be any combination of the 26 letters of the alphabet and the space from the COBOL character set.

NUMERIC

To define an item as numeric, its PICTURE character-string can only contain the symbols 9, P, S, J, and V. Its contents, when represented externally, must be a combination of the numerals 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. A data item defined as numeric may contain a maximum of 23 digits and a maximum of one operational sign. The size restriction of 23 digits does not apply to FILLER items unless the VALUE clause has been specified. If signed, the item may also contain the representation of an operational sign.

ALPHANUMERIC

To define an item as alphanumeric, its PICTURE character-string is restricted to certain combinations of the symbols A, L, X, 9, and the item is treated as if the character-string contained all X's. Its contents, when represented externally, are any of the allowable characters in the COBOL character set. A PICTURE character-string which contains all 9's or all A's, with or without the symbol L, does not define an alphanumeric item.

ALPHANUMERIC EDITED

To define an item as alphanumeric edited, its PICTURE character-string is restricted to certain combinations of the symbols A, X, 9, B, and 0 (zero) given by the following rules:

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

NUMERIC EDITED

To define an item as numeric-edited, its PICTURE character-string is restricted to certain combinations of the symbols B, P, V, Z, 0, 9, , (comma), . (period), *, +, -, CR, DB, and the currency sign ($). The PICTURE character string must contain at least one symbol other than V and 9. The allowable combinations are determined from the order of precedence of symbols and the editing rules. The maximum number of digit positions that may be represented in the character-string is 23.

Classes of Data

The five categories of data items are grouped into three classes: Alphabetic, Numeric, and Alphanumeric. For Alphabetic and Numeric, the classes and categories are synonymous. The Alphanumeric class includes the categories of Alphanumeric Edited, Numeric Edited and Alphanumeric (without editing). Every elementary item belongs to one of the classes and further to one of the categories. The class of a group item is treated at object time as Alphanumeric regardless of the class of elementary items subordinate to that group item. Figure 6-8 depicts the relationship of the class and categories of data items.

LEVEL OF ITEM	CLASS	CATEGORY
Elementary	Alphabetic	Alphabetic
	Numeric	Numeric
	Alphanumeric	Numeric Edited Alphanumeric Edited Alphanumeric
Nonelementary (Group)	Alphanumeric	Alphabetic Numeric Numeric Edited Alphanumeric Edited Alphanumeric

Figure 6-8. Relationship of Class and Category

Function of the Editing Symbols

An unsigned non-zero integer which is enclosed in parentheses following the symbols A, X, 9, P, Z, *, B, 0, +, -, the comma, or the currency sign ($) indicates the number of consecutive occurrences of the symbol. Note that the following symbols may appear only once in a given PICTURE clause: S, J, K, V, L, . (period), CR, and DB.

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

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

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

J The symbol J indicates an operational sign appearing as an overpunch in the most-significant position for DISPLAY and DISPLAY-1 or as a trailing digit in COMP-2. J is not allowed for COMP or COMP-1. J is not counted in the size for DISPLAY or DISPLAY-1 but is counted in COMP-2. Only one operational sign may be present in each PICTURE. J and S are mutually exclusive. See the S sign discussion for the exact bit configuration of signs.

K When the B2500 system dollar option is set, the letter K in a PICTURE character string may be used to indicate the presence of a separate 8-bit character sign appearing in the first character position of an item whose usage is DISPLAY; and is counted in the length of the item. If the usage is COMP-2, the letter K means the same as the letter S. The letter K may be used in a numeric, or edited numeric picture. In both instances, it must be the first character in the PICTURE string. Only one K may be present in the PICTURE string.

L The letter L must appear as the leftmost character in the PICTURE character-string of every elementary item whose length is variable. The class of variable length items may be alphabetic or alphanumeric. The value of data-name determines the actual size of the item and refers to the leftmost characters of the item. The PICTURE designates the maximum size of the item.

P The letter P indicates an assumed decimal scaling position and is used to specify the location of an assumed decimal point when the point is not within the number that appears in the data item. The scaling position character P is not counted in the SIZE of the data item. Scaling position characters are counted in determining the maximum number of digit positions in numeric edited items or NUMERIC

items which appear as operands in arithmetic statements. The scaling position character P can appear only to the left or right as a continuous string of P's within a PICTURE description. Since the scaling position character P implies an assumed decimal point (to the left of P, if P's are leftmost PICTURE characters, and to the right of P, if P's are rightmost PICTURE characters), the assumed decimal point symbol V is redundant as either the leftmost or rightmost character within such a PICTURE description. The character P and the symbol "." (decimal point) cannot both occur in the same PICTURE character string.

S The letter S is used in a character-string to indicate the presence of an operational sign and must be written as the leftmost character in the PICTURE. The S is not counted in determining the SIZE of the elementary item unless USAGE is COMP-2. If USAGE is DISPLAY or DISPLAY-1, S indicates the sign is carried as an overpunch in the least-significant position. J and S are mutually exclusive. For COMP-2, S indicates the sign is carried in the leading digit of the field. The two zone bits of a DISPLAY-1 character are set to 10 when negative, 00 when positive. The four bit sign in EBCDIC and COMP-2 is set to 4"D" for negative, 4"C" for positive.

V The letter V is used in a character-string to indicate the location of the assumed decimal point and may only appear once in a character-string. The V does not represent a character position and, therefore, is not counted in the SIZE of the elementary item. When the assumed decimal point is to the right of the rightmost symbol in the string, the V is redundant.

X Each letter X in the character-string is used to represent a character position which contains any allowable character from the computer's character set.

Z Each letter Z in a character-string may only be used to represent the leftmost leading numeric character positions which will be replaced by a space character when the contents of the character position is zero. Each Z is counted in the SIZE of the item.

9 Each 9 in the character-string represents a character position which contains a numeral and is counted in the SIZE of the item.

0 Each 0 (zero) in the character-string represents a character position into which the numeral zero will be inserted. The 0 is counted in the SIZE of the item.

, Each comma in the character-string represents a character position into which the comma character will be inserted. This character position is counted in the SIZE of the item.

. When the character period appears in the character-string, it is an editing symbol which represents the decimal point for alignment purposes; in addition, it represents a character position into which the period character will be inserted. The period character is counted in the SIZE of the item. For a given program, the functions of the period and comma are exchanged if the clause DECIMAL-POINT IS COMMA is stated in the SPECIAL-NAMES paragraph. In this exchange, the rules for the period apply to the comma and the rules for the comma apply to the period whenever they appear in a PICTURE clause. V and (.) are mutually exclusive.

+, -, CR, DB The symbols +, -, CR, and DB are used as editing sign control symbols. When used, they represent the character position(s) 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 symbol is counted in determining the SIZE of the data-item. (Note that the symbols CR and DB are two character symbols, and any other use of C or D constitutes an error.)

* Each * symbol in the character-string represents a leading numeric character position into which an asterisk will be placed when the contents of that position is zero. Each * is counted in the SIZE of the item.

$ The currency symbol ($) in the character-string represents a character position into which a currency symbol is to be placed. The currency symbol in a character-string is represented by either the dollar sign ($) symbol or by the single character specified in the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph. The currency symbol is counted in the SIZE of the item.

- The symbol - when not the leftmost or rightmost character, is treated as a fixed insertion hyphen. This feature is valid only to the left of the decimal if the preceding character is not the symbol Z.

NOTE

Any other character which is not a defined picture character appearing in the PICTURE is assumed to be an insert character.

Editing Rules

There are two general methods of performing editing in the PICTURE clause: by insertion or by suppression and replacement.

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.

The type of editing which may be performed upon an item is dependent upon the category to which the item belongs. Figure 6-9 specifies which type of editing may be performed upon a given category.

CATEGORY	TYPE OF EDITING
Alphabetic	None
Numeric	None
Alphanumeric	None
Alphanumeric Edited	Simple Insertion, 0 and B
Numeric Edited	All, Subject to Note Above
Any Variable-Length Item	None

Figure 6-9. Permissible Editing Types

INSERTION EDITING

The following are the four types of insertion editing available:

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

SIMPLE INSERTION EDITING. The comma (,), B (space), and 0 (zero) are used as the insertion characters. The insertion characters are counted in the SIZE of the item and represent the position in the item into which the character will be inserted.

While the ANSI74 system dollar option is set, the slash "/" (virgule) may be used as an insert character for alphanumeric edited items. It may be used the same as a "B" or "0" (zero) in the PICTURE character string. Refer to Appendix B for a description of the ANSI 74 implementations.

SPECIAL INSERTION EDITING. The period (.) is used as the insertion character. In addition to being an insertion character, it also represents the decimal point for alignment purposes. The insertion character used for the actual decimal point is counted in the SIZE of the item. The use of the assumed decimal point (represented by the symbol V) and the actual decimal point (represented by the insertion character) in the same PICTURE character-string is prohibited. If the insertion character is the last symbol in the character-string, and the PICTURE clause is not the last clause of the DATA DIVISION entry, the character-string must be immediately followed by the semicolon punctuation character, and then followed by a space. If the PICTURE clause is the last clause of that DATA DIVISION entry, and the insertion character is the last symbol in the character-string, the insertion character must be immediately followed by a period punctuation character followed by a space. This results in two consecutive periods (or ",." if decimal-point is comma has been specified) appearing in the data description entry. 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.

FIXED INSERTION EDITING. The currency sign ($) and the editing sign control symbols "+", "-", CR, and DB are the insertion characters. Only one currency symbol and only one of the editing sign control symbols can be used in a given PICTURE character-string. When the symbols CR or DB are used, they represent two character positions in determining the size of the item, and they must represent the rightmost character positions that are counted in the size of the item. The character "-" may be used as a fixed or floating sign insertion character. When this character appears to the left of the decimal point, its use as either a sign or a hyphen is determined as follows: if the character cannot be legally used as a sign according to the usual rules, then it is interpreted as a hyphen. To the right of the decimal point it is only interpreted as a sign. The symbol "+", when used, must be the leftmost or rightmost character position to be counted in the size of the item. The currency symbol must be the leftmost character position to be counted in the size of the item except that it can be preceded by either a + or a - 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. Depending upon the value of the data item, editing sign control symbols produce the results indicated in figure 6-10.

EDITING SYMBOL IN PICTURE CHARACTER-STRING	RESULT	
	DATA ITEM POSITIVE	DATA ITEM NEGATIVE
+	+	-
-	SPACE	-
CR	2 SPACES	CR
DB	2 SPACES	DB

Figure 6-10. Editing Symbols and Results

FLOATING INSERTION EDITING. The currency symbol and editing sign control symbols + or - are the insertion characters, and they are mutually exclusive as floating insertion characters in a given PICTURE character-string.

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

In the PICTURE character-string, there are only two ways of representing floating insertion editing. One way is to represent any or all of the leading numeric character positions to the left of the decimal point by the insertion character. The other way is to represent all of the numeric character positions in the PICTURE character-string by the insertion character.

The result of floating insertion editing depends upon the representation in the PICTURE character-string. If the insertion characters are only to the left of the decimal point, the result is a single insertion character that will be placed into the character position immediately preceding the decimal point, or the first non-zero digit in the data represented by the insertion symbol string, whichever is farther to the left in the PICTURE character-string.

If all numeric character positions in the PICTURE character-string are represented by the insertion character, the result depends upon the value of the data. If the value is zero, the entire data item will contain spaces. If the value is not zero, the result is the same as when the insertion character is only to the left of the decimal point.

To avoid truncation, the minimum size of the PICTURE character-string for the receiving data item must be the number of characters in the sending data item, plus the number of fixed insertion characters being edited into the receiving data item, plus one for the floating insertion character. When simple insertion characters are embedded within a floating insertion character string, the leftmost two characters of this string must be floating insertion characters. For example, "X PIC $,$$$.99" will produce a syntax error.

SUPPRESSION EDITING. The suppression 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 SIZE of the item. If Z is used, the replacement will be the space, and if the asterisk is used, the replacement character will be the *.

Zero suppression and replacement are 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 associated character position in the data contains a zero. Any of the simple insertion characters embedded in the string of symbols or to the immediate right of this string are part of the string.

In a PICTURE character-string, there are only two ways of representing zero suppression. One way is to represent by suppression symbols, 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 suppression symbols.

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 character positions in the PICTURE character-string are represented by suppression symbols and the value of the data is not zero, the result is the same as if the suppression characters were only to the left of the decimal point. If the value is zero, the entire data item will be spaces if the suppression symbol is Z, or will be all asterisks (*) (except for the actual decimal point) if the suppression symbol is *.

When the asterisk is used as the zero suppression symbol and the clause BLANK WHEN ZERO also appears in the same entry, the zero suppression editing overrides the function of BLANK WHEN ZERO.

REPLACEMENT EDITING. Symbols +, -, *, Z, and the currency symbol, when used as floating replacement characters, are mutually exclusive within a given character-string.

Precedence of Symbols

Figure 6-11 shows the order of precedence when characters are used as symbols in a character-string. An X at an intersection indicates that the symbol(s) at the top of the column may precede, in a given character-string, the symbol(s) at the left of the row. Arguments appearing in braces indicate that the symbols are mutually exclusive. The currency symbol is indicated by the symbol "cs".

At least one of the symbols "A", "X", "Z", "9" or "*", or at least two of the symbols "+", "-" or "cs" must be present in a PICTURE string.

Non-floating insertion symbols "+" and "-", floating insertion symbols "Z", "*", "+", "-", and "cs", and other symbol "P" appear twice in the PICTURE character precedence chart. The leftmost column and uppermost row for each symbol represents its use to the left of the decimal point position. The second appearance of the symbol in the chart represents its use to the right of the decimal point position.

The column and row identified by "/" is to be used for the precedence rules for those other characters which are not defined PICTURE characters but which may be used as fixed insertion characters.

PICTURE Examples

Figure 6-12 shows examples of valid PICTURE clauses based on the precedence of symbols in figure 6-11.

PICTURE

Second Symbol \ First Symbol		Non-Floating Insertion Symbols									Floating Insertion Symbols						Other Symbols						
		B	0	/	,	.	{+ −}	{+ −}	{CR DB}	cs	{Z *}	{Z *}	{+ −}	{+ −}	cs	cs	9	A X	L	S	V	P	P
Non-Floating Insertion Symbols	B	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x			x		x
	0	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x			x		x
	/	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x			x		x
	,	x	x	x	x	x	x			x	x	x	x	x	x	x	x				x		x
	.	x	x	x	x		x			x	x		x		x		x						
	(+ −)																						x
	(+ −)	x	x	x	x	x				x	x	x			x	x	x				x	x	x
	(CR DB)	x	x	x	x	x				x	x	x			x	x	x				x	x	x
	cs						x																
Floating Insertion Symbols	(Z *)	x	x	x	x		x			x	x												
	(Z *)	x	x	x	x	x	x			x	x	x									x		x
	(+ −)	x	x	x	x					x			x										
	(+ −)	x	x	x	x	x				x			x	x							x		x
	cs	x	x	x	x		x								x								
	cs	x	x	x	x	x	x								x	x					x		x
Other Symbols	9	x	x	x	x	x	x			x	x		x		x		x	x		x	x		x
	A X	x	x	x													x	x	x				
	L																						
	S																						
	V	x	x	x	x		x			x	x		x		x		x			x		x	
	P	x	x	x	x		x			x	x		x		x		x			x		x	
	P						x			x							x			x	x		x

Figure 6-11. Order of Precedence

ALPHABETIC ITEMS:
AA
A(25)
ALPHANUMERIC ITEMS:
XX
X(15)
A(5)9(4)
99A99XX
NUMERIC ITEMS:
9
99999
9V99
S99V99
999PPP
J99

EDITED NUMERIC ITEMS (CLASS IS ALPHANUMERIC):	
9.99	
ZZZZZ	
$$.99CR	
B(4)9	
$**,***.99	
-----9	("-" IS A MINUS SIGN)
++,++9.999	
$**,***.99DB	
999,999	
99-99-99	("-" IS A HYPHEN)

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	$$$,$$$.$$	
9(5)	01234	$**,**9.99	$*1,234.00
9(5)	00000	$**,***,**	*******.**
9(5)	00123	$**,**9.99	$***123.00
9(3)V99	00012	$ZZ,ZZ9.99	$ 0.12
9(3)V99	12345	$$$,$$9.99	$123.45
9(3)V99	00001	$ZZ,ZZZ.99	$ .01
9(5)	12345	$$$,$$9.99	$12,345.00
9(5)	00000	$ZZ,ZZZ.ZZ	
9(3)V99	00001	$$$,$$$.$$	$.01
S9(5)	(+) 12345	ZZZZ9.99+	12345.00+
S9(5)	(-) 00123	--99999.99	-00123.00
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
S9(5)	(-) 123	-ZZZZZ.99	- 123.00

Figure 6-12. PICTURE Clause Examples

RANGE

The RANGE clause indicates the potential range of a data item. This clause is used for documentation purposes only.

The format for this clause is:

```
RANGE IS literal-1 {THROUGH} literal-2
                   {THRU   }
```

The RANGE clause can be written only at the elementary item level.

RECEIVED

The RECEIVED clause identifies those items which are received as parameters by name or by value from another procedure or items which are to be passed to another procedure by name or by value. The RECEIVED clause may only apapear on a level 77 or level 01 in a LOCAL-STORAGE or WORKING-STORAGE section.

The format is as follows:

$$\text{RECEIVED BY } \left\{\begin{matrix}\underline{\text{REFERENCE}}\\ \underline{\text{REF}}\\ \underline{\text{CONTENT}}\end{matrix}\right\}$$

The RECEIVED BY REFERENCE allows two or more procedures to share the item with which it appears. Any reference to the identifier in one of the procedures which shares it will be referencing the same common data area as the others use. This is also called passing data "by name". REF is synonymous with REFERENCE. Parameters passed "by value" are identified by the clause RECEIVED BY CONTENT. In this case the current value of the identifier is received by the procedure. Though another procedure may change the value it has associated with that data-name, it merely affects its own copy of its storage; likewise, the receiving procedure may make changes to data-name that will not affect the original item.

The RECEIVED BY CONTENT clause may not appear with any item whose usage is described as CONTROL-POINT, EVENT or LOCK or with any item described at the 01 level.

When RECEIVED is not specified, 77 level COMP-1 items are RECEIVED BY CONTENT and all other items and files are RECEIVED BY REFERENCE.

A data description entry containing the RECEIVED clause must not contain a VALUE clause.

Figure 6-13 is an example of use of the RECEIVED clause in a calling and a called program.

```
SPECIAL-NAMES.  "CALLED"/"PROGRAM" IS TO-BE-CALLED.
FILE-CONTROL.
    SELECT LOCAL REF DIOFL ASSIGN DIRECT DISK.
    SELECT PRTF ASSIGN DIRECT DISK.
    SELECT SHOWIT ASSIGN PRINTER.
DATA DIVISION.
FILE SECTION.
FD PRTF RECORD CONTAINS 132 DISPLAY CHARACTERS.
FD DIOFL RECORD CONTAINS 132 DISPLAY.
FD SHOWIT.
01 DISDIR PIC X(132).
01 ITSHEADING.
    03 EMPTYCMNT PIC X(132).
WORKING-STORAGE SECTION.
77 VAL PIC 99 COMP-1.
77 DONE EVENT.
77 TSK USAGE IS CONTROL-POINT.
01 TBL.
    05 COL OCCURS 5 PIC X.
01 SET-UP SIZE 132 DISPLAY RECORD AREA.
    03 FILLER VALUE "MAIN" PIC X(7).
    03 TBLVAL PIC 9(5) VALUE ZEROS.
    03 FILLER VALUE SPACES PIC X(120).
01 FROMHERE.
    03 CMNT PIC X(132) VALUE "CONTENTS OF DISK FILE".
01 PLAINLINE RECORD AREA SIZE 132.
LOCAL-STORAGE SECTION.
LD FORPARAMS.
77 VAL-A PIC 99 COMP-1 RECEIVED BY CONTENT.
77 FINE REF EVENT.
01 ELBAT SIZE 5 REF DISPLAY.
    03 NUMB PIC X(5).

PROCEDURE DIVISION.
DECLARATIVES.
S1 SECTION.  USE EXTERNAL TO-BE-CALLED AS PROCEDURE
    ; WITH FORPARAMS, DIOFL
    ; USING VAL-A, FINE, ELBAT, DIOFL.
END DECLARATIVES.
```

Figure 6-13. RECEIVED Clause in Calling and Called Programs.

```
P1.
    OPEN OUTPUT PRTF.
    MOVE 1 TO VAL.
P3.
    WRITE PRTF KEY IS VAL  FROM SET-UP USING DONE.
    CALL TSK WITH S1
        USING VAL, DONE, TBL, PRTF.
XIT.
    ADD 2 TO VAL.
    MOVE TBL TO TBLVAL.
    WRITE PRTF KEY IS VAL FROM SET-UP USING DONE.
    WAIT SET-UP.  RESET DONE.
SHOW-FILE.
    CLOSE PRTF.
    MOVE ZERO TO VAL.
    OPEN OUTPUT SHOWIT.
    MOVE FROMHERE TO EMPTYCMNT.
    WRITE ITSHEADING.
    OPEN INPUT PRTF.
WRITE-ONE.
    ADD 1 TO VAL.
    READ PRTF KEY IS VAL INTO PLAINLINE USING DONE.
    WAIT DONE.
    RESET DONE.
    WRITE DISDIR FROM PLAINLINE.
    IF VAL GREATER THAN 2 GO STOPPING
        ELSE GO WRITE-ONE.
STOPPING.
    STOP RUN.
```

RECEIVED

```
FILE-CONTROL.
     SELECT PRTFIL ASSIGN DIRECT DISK.
DATA DIVISION.
FILE SECTION.
FD PRTFIL RECORD CONTAINS 132 DISPLAY.
WORKING-STORAGE SECTION.
77 VAL-A PIC 99 COMP-1 RECEIVED BY CONTENT.
77 DONE EVENT REFERENCE.
01 ELBAT REF PIC X(5).
01 ARRIVAL RECORD AREA SIZE 132.
     03 MYNAME PC X(7) VALUE "TASK".
     03 TABVAL PIC X(5).
     03 FILLER VALUE SPACES PIC X(120).
PROCEDURE DIVISION
     USING VAL-A, DONE, ELBAT, PRTFIL.
P1.
     WAIT DONE.
     MOVE "12345" TO ELBAT.
     MOVE ELBAT TO TABVAL.
     ADD 1 TO VAL-A.
     WRITE PRTFIL KEY IS VAL-A FROM ARRIVAL USING DONE.
     GO TO P1.
```

Results of Executing Calling Program:

```
     CONTENTS OF DISK FILE
         MAIN 00000
         TASK 12345
         MAIN 12345
```

RECORD AREA

The RECORD AREA clause specifies that the record being described is to be used for direct I/O buffering. This clause may only appear on the 01 level in a WORKING-STORAGE SECTION or a LOCAL—STORAGE SECTION.

Areas described with the RECORD AREA clause will become non-overlayable until the area is specified in a DEALLOCATE statement.

An area described with the RECORD AREA clause must not be declared to be COMP-1.

See example in RECEIVED clause discussion for an example of using direct files and record areas.

REDEFINES

The REDEFINES clause allows the same area of memory to be referenced by more than one data-name with different formats and sizes.

The format for this clause is as follows:

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

The REDEFINES clause pertains to only part of a record in the FILE SECTION. Implicit redefinition is provided by the DATA RECORDS clause in the file description entry. REDEFINES may apply to a part or all of a record in the CONSTANT SECTION or WORKING-STORAGE SECTION. The rules listed below must be followed when using this clause:

a. The REDEFINES clause, when specified, must immediately follow data-name-1.
b. The level-numbers of data-name-1 and data-name-2 must be identical, but must not be 66 or 88. Qualification of data-name-2 is never required.
c. When REDEFINES is applied to other than an 01 level, the area being redefined must be the same size as the area of the new description. A FILLER entry may be used to avoid violating this rule.
d. Redefinition ceases whenever a level-number less than or equal to that of data-name-1 or data-name-2 is encountered.
e. Multiple redefinitions of the same storage area are permitted. The entries giving the new descriptions of the storage area must follow the entries defining the area being redefined, without intervening entries that define new storage areas. Multiple redefinitions of the same storage area must all redefine the data-name of the entry that originally defined the area.
f. The entries giving the new description of the storage area must not contain any VALUE clauses, except in condition-name entries.
g. A level 01 data item with an OCCURS clause may not have a REDEFINES clause.

h. The REDEFINES clause, when specified, may REDEFINE a DISPLAY data item with a COMP-2 data item whose internal size is four bits less than the internal size of the DISPLAY data item. The redefining item will reference the left-most part of the DISPLAY item. This application is compatible with Burroughs medium and small systems COBOL.

i. A REDEFINES clause can optionally specify an immediately-preceding redefining data item of the same hierarchical level, rather than the data item originally defining the area.

NOTE: Rules (h) and (i) apply only while the B2500 system dollar option is set.

Figure 6-14 shows examples of redefinition.

```
01 WORK1  SZ  80.
   02  PART-ONE  SZ  60.
   02  PART-TWO  REDEFINES  PART-ONE.
       03  XX  SZ  40.
       03  YY  SZ  20.
   02  PART-THREE REDEFINES PART-ONE.
       03  ZZ  SZ  55.
       03  FILLER  SZ  5.
   02  FILLER  SZ  20.

01 TBL-1  SZ 24.
   03  AA  PIC  X  OCCURS 0 TO 24 TIMES
       DEPENDING ON DN.
01 TBL-2  REDEFINES  TBL-1.
   03  BB  PIC  XX  OCCURS 0 TO 12 TIMES
       DEPENDING ON DN-2.

01  AA1.
   05  AA2.
       10  AA3  COMP-2  PIC  S999.
       10  AA4  COMP-2  PIC  S9(5).
   05  BB2  REDEFINES AA2.
       10  BBX  PIC  X(5).
   05  FILLER  SIZE 61.
```

Figure 6-14. Examples of REDEFINES

RENAMES

The RENAMES clause permits alternative, and possibly overlapping grouping of elementary items.

The following is the format for the RENAMES clause:

66 data-name-1; RENAMES data-name-2 [{ THROUGH / THRU } data-name-3]

One or more RENAMES entries can be written for a logical record. All RENAMES entries associated with a given logical record must immediately follow its last record description entry. It is not possible to "chain" RENAMES; i.e., it is illegal to rename data item "A" to "B" and then rename "B" to "C". However, multiple RENAMES of a data-name are permitted. (See example below.)

Data-name-2 and data-name-3 must be names of elementary items or groups of elementary items of the same logical record and cannot be the same data-name. A 66 level entry cannot rename another 66 level entry nor can it rename a 77, 88, or 01 level entry.

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.

The beginning of the area described by data-name-3 must not be to the left of the beginning of the area described by data-name-2. The end of the area described by data-name-3 must not be to the left of the end of the area described by data-name-2. Data-name-3 cannot be contained within data-name-2. Data-name-2 and data-name-3 may be qualified.

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

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

Data-name-1 will assume the USAGE of the item being renamed. If the THRU option is used, all items within the RENAMES range must have the same USAGE.

Figure 6-15 shows examples of the RENAMES clause.

```
01 TAB SIZE 24.
    03 A SIZE 8.
        05 A1 PIC X.
        05 A2 PIC XXX.
        05 A3 PIC XX.
        05 A4 PIC XX.
    03 X SIZE 16.
        05 X1 PIC XX.
        05 X2 PIC X(6).
        05 X3 PIC X(8).

66 B RENAMES A.              (i.e.,   A1 THRU A4)
66 C RENAMES A.              (i.e.,   A1 THRU A4)
66 D RENAMES A1 THRU A3.
66 E RENAMES A4 THRU X2
66 F RENAMES A2 THRU X.      (i.e.,   A2 THRU X3)
66 G RENAMES A THROUGH X.    (i.e.,   A1 THRU X3)
```

Figure 6-15. Examples of RENAMES

SEGMENT

The SEGMENT clause may be used to take advantage of the segmented array hardware for very large string arrays whose entire presence in core is not required.

The reserved word SEGMENT at the 01 level in WORKING or LOCAL STORAGE declares that the array will be segmented.

The SEGMENT clause may be used with 01 items whose usage is ASCII, DISPLAY, DISPLAY-1, COMP, COMP-2, COMP-4 or COMP-5. Redefinitions of these arrays are also considered to be segmented, though it is not necessary that the SEGMENT clause be used.

01 items of COMPUTATIONAL usage whose size is more than 1023 words are implicitly segmented, along with any redefinitions of this array.

While core usage requirements are decreased, there is a tradeoff. References to items within segmented arrays require an extra index operation in the hardware.

SIZE

The SIZE clause is used to specify the number of characters in a data item.

The format for the SIZE clause has two options as follows:

Option 1:

```
{SIZE}
{SZ  }  IS integer-2 CHARACTERS
```

Option 2:

```
{SIZE}
{SZ  }  IS [integer-1 TO] integer-2 CHARACTERS
        [DEPENDING ON data-name-1]
```

Integer-1 and integer-2 must be positive integers. When both are used, integer-1 must be less than integer-2. The data description of data-name must be such that it defines a positive integer.

Any one of the key words of the USAGE clause can be inserted between integer-2 and the reserved word CHARACTERS.

The number of characters in every elementary item must be specified by means of the SIZE or PICTURE clauses, unless the USAGE of the item is COMP-4, COMP-5, CP, EVENT, LOCK or INDEX. PICTURE and SIZE need not both be given for an item; however, if both are given, the PICTURE is used in determining the SIZE of the item.

When option 1 is used, integer-2 specifies the exact number of characters excluding the operational sign.

Option 2 specifies that the item is of variable length. The DEPENDING ON clause is not required, but if used, integer-1 is ignored.

The value of data-name-1 is the count of the number of characters in the item being described. Its value must not exceed integer-2 and must not be less than one.

If data-name-1 appears in the records of a file, its least-significant character position must always be the same number of character positions from the beginning of each record.

SIZE

For fixed-length group items, the SIZE clause is for documentation purposes only. The size of a group is determined from the sum of the number of characters in the elementary items subordinate to that group. When the size specified at the group level does not equal the sum of the lengths of its subordinate elementary items, the compiler will issue a warning message. This facility is useful for checking the sizes of lengthy or complicated groups.

In option 2, the maximum size of a group is the sum of the number of characters in the fixed-length elementary items plus the sum of the maximum number of characters in the variable-length items.

In option 2, if any item within a group contains the OCCURS clause, the maximum size of the table is the product of the number of characters in the item times the maximum number of occurrences.

If option 2 is used, the number of characters moved is the value of the DEPENDING ON data-name. If the DEPENDING ON clause is omitted, the maximum size is moved.

If option 2 is used on an 01 level in the FILE SECTION, the clause is used only to determine the initial setting of the attribute FILETYPE.

The number of characters transferred by group moves involving one or more groups of variable lengths is derived from the logical size of the groups as specified by the value of the depending data-name or first four characters of the group, whichever is appropriate.

For example:

```
FILE SECTION.

FD TAPE-OUPT; RECORD CONTAINS 12 TO 92 CHARACTERS.
01 TAPEREC SIZE 12 TO 92 DISPLAY-1.
    02 SF PIC 9(4)
    02 COMMENT SIZE 8.
    02 DDTA PIC X(80).

WORKING-STORAGE SECTION.
01 RECOUT SIZE 20 TO 192 DEPENDING ON SIZ.
    03 SIZ PIC 9(4)
    03 CMT PIC X(8).
    03 DAT PIC X(180).
         .
         .
         .
PROCEDURE DIVISION.
         .
         .
         .
    MOVE RECOUT TO TAPEREC.
```

The statement MOVE RECOUT TO TAPEREC will transfer the number of characters specified by the value of SIZ in RECOUT, from RECOUT to TAPEREC. If the value of SF (the first four characters of TAPEREC) is less than or equal to the value of SIZ, then SF characters are transferred to TAPEREC. If the value of SF exceeds the value of SIZ, then SIZ characters are transferred to TAPEREC, followed by (SF-SIZ) space fill of the remaining character positions.

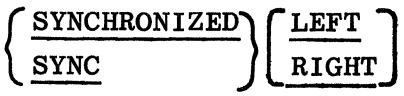
Note in the above example that the DEPENDING clause has not been specified for TAPEREC. Since variable-length records are being written, the first four characters of the record are used as the DEPENDING item.

Before any access is made to TAPEREC, SF must be properly initialized, SIZ must be properly initialized prior to any reference to the data-name RECOUT.

SYNCHRONIZED

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

The following is the format for this clause:



This clause may only appear with an elementary item. It may not appear with any items described as CONTROL-POINT, EVENT, INDEX or LOCK. It specifies that the COBOL 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 (explicit or implicit) 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 size of any group to which the elementary item belongs, and
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 such that it will begin at the left boundary of a computer word.

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

When SYNCHRONIZED is specified without the words LEFT or RIGHT, then RIGHT is assumed.

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

When the SYNCHRONIZED clause is specified for an item within the scope of an OCCURS clause, an exception to the above rules will be in effect in that each occurrence of the item is SYNCHRONIZED to a character boundary.

An item with USAGE of COMP, COMP-4, or COMP-5 need not begin at a word boundary. However, if the SYNCHRONIZED clause is used with such an item, more efficient code will result. Synchronization of COMP, COMP-4 or COMP-5 items may be done either SYNC RIGHT or SYNC LEFT, since all such items use a full computer word (double-precision uses two full words).

USAGE

The function of the USAGE clause is to specify the format in computer storage of a data item.

```
[USAGE IS] { COMP                 { ASCII
           { COMPUTATIONAL        { DISPLAY
           { COMP-1               { DISPLAY-1
           { COMPUTATIONAL-1      { INDEX
           { COMP-2               { EVENT
           { COMPUTATIONAL-2      { LOCK
           { COMP-4               { CONTROL-POINT
           { COMPUTATIONAL-4      { CP
           { COMP-5
           { COMPUTATIONAL-5
           { INDEX FILE CONTAINS file-name-1 [,file-name-2]...
```

In the absence of any USAGE indication, either explicitly shown in the USAGE clause or within the SIZE clause, USAGE IS DISPLAY is assumed.

The USAGE clause for a report group item can only specify the DISPLAY or DISPLAY-1 option.

The USAGE clause can be written at any level, except CONTROL-POINT, EVENT, INDEX and LOCK may only appear at level 77 or level 01 in WORKING-STORAGE or LOCAL-STORAGE. If the USAGE clause is written at a group level, it applies to each elementary item in the group. Multiple record descriptions for the same file may be declared with contradictory usages. If a group item is described as COMPUTATIONAL, the elementary items are COMPUTATIONAL. The group item itself is not COMPUTATIONAL because it cannot be used in computations.

COMP, COMP-2, COMP-4 and COMP-5 items should not be declared for DISPLAY files (CARD-READER, PRINTER, PUNCH, or REMOTE). The use of such items will cause question marks to print for characters which have no graphic and may cause undesired control characters to be developed.

Care should be exercised in redefining COMP, COMP-1, COMP-4, COMP-5 items on a character basis since the decimal number does not correspond in mapping to its binary equivalent.

The following rules apply when mixing usages:

1. COMP-2 items cannot be subordinate to ASCII, COMP, COMP-4, COMP-5 or DISPLAY-1 group items.
2. COMP, COMP-4 and COMP-5 items can be subordinate to COMP, COMP-1, COMP-4, COMP-5, DISPLAY or DISPLAY-1 group items.

3. COMP-2 items can be subordinate to DISPLAY group items. The compiler, however, may have to insert a 4-bit filler if DISPLAY elementary items follow COMP-2 elementary items.
4. Extreme care must be exercised when moving DISPLAY or DISPLAY-1 group items if either the sending or receiving field contains subordinate COMP, COMP-2, COMP-4 or COMP-5 data.
5. No mixing of USAGE or contradiction of USAGE is permitted subordinate to a group declared CONTROL-POINT, EVENT, INDEX or LOCK. These items may not be subordinate to an item of any other USAGE.
6. Automatic translation of data takes place when moving data between ASCII, COMP-2, DISPLAY, and DISPLAY-1 items. Any character in one character set which is undefined in the other character set is translated into the question mark.
7. INDEX, EVENT, LOCK and CP items must not be declared subordinate to an item of USAGE COMP, COMP-1, COMP-2, COMP-4, COMP-5, ASCII, DISPLAY or DISPLAY-1. INDEX, EVENT, LOCK and CP items may not be mixed in an 01 level. When an array is to contain INDEX, EVENT, LOCK or CP items, USAGE must be declared at the 01 level.
8. Floating-point format numeric literals may only be moved to floating-point COMP-4 or COMP-5 receiving fields.

USAGE is the dominant declaration for internal format representation within the computer system and is defined for use as follows:

a. DISPLAY. The data item is assumed to contain eight-bit-coded EBCDIC characters, six characters for each B 7000/B 6000 word.
b. DISPLAY-1. The data is assumed to contain six-bit-coded BCL characters, eight characters per computer word.
c. ASCII. The data is assumed to contain eight-bit-coded ASCII characters, six characters for each B 7000/B 6000 word. If an array is to contain ASCII characters, the 01 level must be declared ASCII and no subordinate item may declare a usage other than ASCII. No data item within an ASCII array may be declared as numeric or edited numeric.
d. COMPUTATIONAL (COMP). The data item is to be used primarily for arithmetic operations; therefore, it is maintained in a binary coded representation.

    1. COMPUTATIONAL implies a signed item. No further sign specification is required. (An unsigned value may be obtained through the ABS function.)

2. A COMPUTATIONAL item is capable of representing a value to be used in computations and therefore is always numeric. Only elementary items may be used in computations.
3. COMP items will occupy memory as follows:

   When the declared size is less than or equal to 11 decimal digits, the actual size is equal to one computer word; however, the item is not necessarily word-aligned. (This is equivalent to six DISPLAY digits or eight DISPLAY-1 digits.)

   When the declared size is greater than 11 digits, the actual size is equal to two computer words (the equivalent of 23 decimal digits); however, the item is not necessarily word-aligned.

   The actual size is used when determining the size of a record, and for testing for size error conditions.
4. When the B2500 system dollar option is set, USAGE COMPUTATIONAL will be considered to mean four bit character data (COMP-2).

   Four-bit character elementary data items can be greater than 23 characters in length while the B2500 system dollar option is set, provided they are declared as unsigned and as having no scaling positions. Such items are not allowed to be used in an arithmetic statement.
5. The PICTURE of a COMPUTATIONAL item can only contains nines, the operational sign character "S"; the implied decimal point character "V"; and one or more of the character "P".
6. COMP items are not required to start at a word boundary, although faster execution will result when COMP items start at a word boundary.
7. If the ANSI74 or USASI dollar option is set at the time a COMPUTATIONAL data item is declared, the item will be considered throughout the program to be capable of receiving only values less than or equal to the maximum decimal number specified by the PICTURE clause.

   For example, given the following item, the maximum value that can be stored in X is 99:

```
77 X COMP PIC 99
```

NOTE: If no sign is specified in the picture clause, only absolute values are stored in the item. An attempt to store a larger number than the PICTURE specifies will cause a size error condition.

If strict compliance with the ANSI 74 standard is sought, it is recommended that either the USASI or ANSI74 system dollar options be set. Refer to Appendix B for a description of the ANSI 74 implementations.

e. COMPUTATIONAL-1 (COMP-1). For a 77 level item, COMPUTATIONAL-1 is synonymous with COMPUTATIONAL. The above discussion of COMPUTATIONAL applies to COMPUTATIONAL-1 items, with the exception of d-4, d-7 and that COMP-1 items may be declared only on level 77 or 01 entries (or on the individual items within a COMP-1 array) in WORKING-STORAGE and LOCAL-STORAGE.

COMP-1 items, when used in arithmetic operations, do not achieve any substantial reduction in execution time over COMPUTATIONAL items.

An 01 level described as COMP-1 becomes a stack array, thus speeding up access to the array. Subordinate items must be either COMP, COMP-1, COMP-4 or COMP-5. The 01 item may be redefined and the new item declared DISPLAY, DISPLAY-1, ASCII, COMP, COMP-1, COMP-2, COMP-4 or COMP-5. Neither a stack array nor its redefinition may be passed as parameters or referenced as globals.

Stack (COMP-1) arrays may not be declared with the GLOBAL, LOWER-BOUND, OCCURS, OWN, RECORD AREA or SEGMENT clauses at the 01 level.

f. COMPUTATIONAL-2 (COMP-2). The data item is assumed to contain hexadecimal digits, 12 digits per computer word. COMP-2 items need not begin at the byte boundary or word boundary nor is a sign required. Regardless of sign specification, if the left-most four bits of a COMP-2 item contain a value greater than 9, then these four bits are treated as a sign. If a COMP-2 item is described as "PIC 99", and contains a hex value "A5", then three four-bit characters are accessed (i.e., a sign and two digits). If a COMP-2 item described as "PIC S99" contains the hex value "567", the value "56" will be used.

g. COMPUTATIONAL-4 (COMP-4). This data item will be a single precision internal floating point operand which will occupy one word of memory; however, the item is not necessarily word-aligned.

   A PICTURE entry is not permitted for a COMP-4 item.

   All comments for COMP items except the size requirements and allowable PICTURE characters apply equally to COMP-4 items.

h. COMPUTATIONAL-5 (COMP-5). This data item will be a double precision internal floating point operand which will occupy two words of memory; however, the item is not necessarily word-aligned.

   A PICTURE entry is not permitted for a COMP-5 item.

   All comments for COMP items except the size requirements and allowable PICTURE characters apply equally to COMP-5 items.

i. INDEX. An elementary item described with the USAGE IS INDEX clause is called an index data item. If a group item is described with the USAGE IS INDEX clause, the elementary items in the group are all index data items. The group itself is not an index data item and cannot be used in SEARCH or SET statements or in a relational condition. An index data item can be referred to directly only in a PERFORM, SEARCH, or SET statement, or in a relational condition. An index data item can be part of a group referred to in a MOVE or input-output statement, in which case no conversion will take place. The SIZE,SYNCHRONIZED, BLANK WHEN ZERO, JUSTIFIED, PICTURE, and VALUE clauses cannot be used to describe group or elementary items described with the USAGE IS INDEX clause.

   Index data items can also be declared subordinate to a usage DISPLAY group item. This offers compatibility with both B 4700 and B 1700 systems COBOL languages and with the ANSI 1968 and ANSI 1974 COBOL standards. The internal representation of all index data names is that of a signed seven-digit four bit character field. This internal representation is the same as the B 4700 and B 1700 COBOL compilers give for index data items.

j. INDEX FILE. DIRECT SWITCH FILES may be declared at the 77 level in WORKING or LOCAL-STORAGE SECTIONS by specifying a usage of INDEX FILE.

Example:

```
77 switch-file-identifier INDEX FILE [CONTAINS file-name-1
     [,file-name-2]  ...]
```

The CONTAINS clause is used to describe which DIRECT files compose the switch. Each file named in the switch must be a DIRECT file and an FD must be provided.
The CONTAINS clause must be present if the DIRECT SWITCH FILE is not received as a parameter. When the DIRECT SWITCH FILE is received as a parameter, or declared in LOCAL-STORAGE, then the RECEIVED clause must be used to indicate that it is RECEIVED BY REFERENCE (name) and the CONTAINS clause must not be specified.

A DIRECT SWITCH FILE identifier can be used any place in the syntax that a DIRECT FILE identifier can be used, namely in OPEN, CLOSE, READ, and WRITE statements and in attribute expressions.

Example:

```
OPEN INPUT SWFL(X)
:
     READ SWFL(X) KEY IS RCINR(X) INTO RCDAREA(X)
```

All reads and writes with DIRECT SWITCH FILES must use a KEY clause if non-serial action is desired, even if all the DIRECT FILES in the switch are declared to be random.

If DIRECT SWITCH FILES are passed as parameters, then the corresponding formal parameter description must be a DIRECT SWITCH FILE. A program which receives a DIRECT SWITCH FILE as a parameter must not have an FD for the files contained in the switch, since these files were described in the program which passed them as parameters.

k. EVENT. Items described with the USAGE IS EVENT clause are used to give the programmer a means of testing and controlling DIRECT input-output operations (i.e., I-O COMPLETE). For asynchronous processing, EVENT's may be used as a common interlock between two or more processes thus providing an efficient means of correlating the activities of one process with its related processes.

EVENT usage is allowed only on a 77 or 01 level item and, if used at an 01 level, may have a subordinate OCCURS clause. (See the OCCURS clause.) Except for documentary uses of the SIZE clause, no other entries are permitted with an EVENT name.

Elementary EVENT items occupy two words of memory.

For information and syntax for controlling and testing EVENT names, see the following PROCEDURE DIVISION statements: READ and WRITE options for DIRECT I-O, CAUSE, RESET, IF, and WAIT.

l. LOCK. When a data usage is declared as LOCK, the same rules apply as those declaring data usage as EVENT. For information and syntax for controlling and testing LOCK names, see the LOCK and UNLOCK statements. Elementary LOCK items occupy two words of memory.

m. CONTROL-POINT (CP). An elementary item implicitly or explicitly described with a USAGE IS CONTROL-POINT clause is known as a control-point item. The elementary item cannot be a conditional variable. If a group item is described with the USAGE IS CONTROL-POINT clause, the elementary items in the group are all control-point items. The group itself is not a control-point item and may not be used in any statement except the USING phrase or within a parameter list.

Elementary CP items are data descriptors and as such, occupy a single word of memory.

An elementary CONTROL-POINT can be referred to directly only in an ATTACH, CALL, DETACH, RUN, EXECUTE, PROCESS, or SET statement, or in a USING phrase, task attribute expression, or in a parameter list. Further explanation of CONTROL POINT items may be found in the descriptions of statements which reference them and in the task attribute descriptions.

VALUE

The VALUE clause can serve two functions. First, it specifies the values for constants and the initial values for WORKING-STORAGE items. Second, it defines the values or range of values associated with a condition name.

The format of the VALUE clause consists of two options:

Option 1:

```
{VA   }
{VALUE} IS literal
```

Option 2:

```
{VA    }          [IS ]
{VALUE }          [ARE]  literal-1 [{THROUGH} literal-2]
{VALUES}                          [{THRU   }          ]

    [,literal-3 [{THROUGH} literal-4]] ...
                 {THRU   }
```

Option 1 serves the first function; option 2 serves the second. The VALUE clause cannot be stated for any item whose size, explicitly or implicitly, is variable.

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

a. If the category of the item is numeric, all literals in the VALUE clause must be numeric literals. If the literal defines the value of a WORKING-STORAGE or CONSTANT item, the literal is aligned according to the alignment rules, except that the literal must not have a value which would require truncation of non-zero digits.

b. If the category of the item is alphabetic or alphanumeric, all literals in the VALUE clause must be non-numeric literals. The literal is aligned according to the alignment rules (refer to the JUSTIFIED clause) except that the number of characters in the literal must not exceed the size of the item.

c. The numeric literals in a VALUE clause of an item must have a value which is within the range of values indicated by the SIZE or PICTURE clause; for example, for PICTURE PPP99, the literal must be within the range .00000 thru .00099.

d. The functions of the editing characters in a PICTURE clause have no effect on initialization of the item. The VALUE clause is the only clause that may (depending on its usage) provide initialization. Editing characters are included, however, in determining the size of the item.

A figurative constant may be substituted for literal.

Rules governing the use of the VALUE clause differ with the respective section of the DATA DIVISION as shown below:

a. In the FILE SECTION, unless the B2500 system dollar option is set, the VALUE clause may be used only in condition-name entries.
b. In the WORKING-STORAGE section, the VALUE clause must be used in condition-name entries, and it may also be used to specify the initial value of any data item. The VALUE clause causes the item to assume the specified value at the start of the object program. If the VALUE clause is not used in the description of the item, the initial value is unpredictable.
c. In the CONSTANT SECTION, the VALUE clause must be used to specify the value assumed by each constant data item in the object program.

The VALUE clause must not be stated in a data description entry that contains an OCCURS clause, or in an entry subordinate to an entry containing an OCCURS clause. This does not apply to condition-name entries.

The VALUE clause cannot be used to describe any item whose usage is EVENT, INDEX, LOCK, or CONTROL-POINT.

The VALUE clause must not be stated in a data description entry that contains a REDEFINES clause, or in an entry that is subordinate to an entry containing a REDEFINES clause. This does not apply to condition-name entries.

If the VALUE clause is used in an entry at the group level, the literal must be a figurative constant or a non-numeric literal, and the group area is initialized without consideration for the individual elementary or group items contained within this group. The VALUE clause cannot be stated at the subordinate levels within the group. Caution should be used in specifying initial values for group items containing embedded COMP or COMP-2 data items.

WORKING-STORAGE SECTION

Concept of WORKING-STORAGE

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

All working storage areas not assigned a specific value will have an unpredictable initial value.

Organization

Whereas the FILE SECTION is composed of file description and record description entries, the WORKING-STORAGE SECTION is composed of non-contiguous working-storage and record description entries (contiguous working-storage). The WORKING-STORAGE SECTION begins with a section-header and a period, followed by item description entries for non-contiguous working-storage items, intermixed with record description entries for working-storage records.

The format for the 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-P
             88 CONDITION-NAME-2

77 DATA-NAME-Q
01 DATA-NAME-R
```

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 is defined in a separate entry which begins with the special level number 77. The following data description clauses are required in each entry:

a. Level-number.
b. Data-name.
c. SIZE or PICTURE (if the item is not an EVENT, LOCK, or CP).

WORKING-STORAGE Records

Items in WORKING-STORAGE which bear a definite relationship to one another may be grouped into records according to the rules for formation of record descriptions. All record description clauses can be used in a WORKING-STORAGE record description. Each WORKING-STORAGE record-name (01 level) must be unique since it cannot be qualified by a file-name or section-name. Subordinate data-names need not be unique if they can be made unique by qualification when referenced.

Initial Values

The initial value of any item in the WORKING-STORAGE SECTION is specified by using the VALUE clause. If VALUE is not specified, the initial values are unpredictable. The initial value of any index data name is determined at compile time.

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, since the VALUE clause serves two different purposes.

Coding the WORKING-STORAGE SECTION

Figure 6-16 illustrates the coding of the WORKING-STORAGE SECTION.

Burroughs COBOL CODING FORM

PROGRAM: WORKING-STORAGE SECTION Coding | REQUESTED BY | PAGE 1 OF 1
PROGRAMMER: KIMBERLY | DATE 25 SEPT | IDENT. 73 80

```
PAGE NO. 1 3  LINE NO. 4 6  7  A 8 11  B 12 72
01  WORKING-STORAGE SECTION.
02  77  PAGE-COUNTING                                  PIC 9(4).
03  01  PRINT-HEADING.  05  FILLER  SIZE 20 VALUE SPACES.
04  05  FILLER  VALUE "M O N T H L Y  A C C O U N T S  P A Y A B L
05  -       "E  R E P O R T"                            SIZE 120.
06  77  GRAND-TOTAL                  COMP-1            PIC 9(8) V99.
07  77  SEQ-HOLD                     COMP-2            PIC 9(6).
08  01  HOLD-INDEX                   INDEX.
09      05  INDEX-1.
10      05  SOME-MORE-INDICES        OC 10.
11
12  77  MONTH-SUB                    COMP              PIC 99.
13  01  MTBL  VA "JANFEBMARAPRMAYJUNJULAUGSEPOCTNOVDEC"  SIZE 36.
14      05  MONTH-NAME               OCCURS 12         PIC XXX.
15
16  01  TOTALS-TABLE.
17      05  LIQUIDS                  OCCURS 100 TIMES.
18          10  DESCRIPTIONS                           PIC X(10).
19          10  QTYS                 COMP-2            PIC 9(4).
20      05  OTHERS  OC 3.  10 SOLIDS OC 25.
          15  METALS                 OC 10.
            20  TYPE                                   PIC XX.
            20  SHIPMENTS            OCCURS 10.
              29  DATE-OF-SHIPMENT                     PIC 9(5).
              29  QTY                                  PIC 9(3).
```

Figure 6-16. WORKING-STORAGE SECTION Coding

CONSTANT SECTION

Concept of Constant Storage

The concept of literals and figuratives enables the user to specify the value of a constant by writing its actual value (or a figurative representation of that value). It is often desirable to name this value and then refer to it by its name. For example, 6% (.06) may be named as INTEREST-RATE and then referred to by its name (INTEREST-RATE), instead of its value (.06). These named constants may be declared in the CONSTANT SECTION under the same general format as data-names declared in the WORKING-STORAGE SECTION. However, constants so declared may be used but not altered by the program; hence, such storage differs from WORKING-STORAGE.

Organization

The CONSTANT SECTION is organized in exactly the same way as the WORKING-STORAGE SECTION, beginning with a section header and a period, followed by item description entries for non-contiguous constant items, and then by record description entries for constant storage records and their subordinate entries. The skeletal format for the CONSTANT SECTION is as follows:

```
CONSTANT SECTION.
77 DATA-NAME-1
.
.
.
77 DATA-NAME-N
01 DATA-NAME-2
    02 DATA-NAME-3
    .
    .
    .
01 DATA-NAME-4
    02 DATA-NAME-5
        03 DATA-NAME-6
```

Non-Contiguous Constant Storage

Constants which bear no relationship to one another are classified and defined as non-contiguous constants. Each of these constants is defined in a

separate item description entry which begins with the special level number 77. The following description clauses are required in each entry:

a. Level-number.
b. Data-name.
c. SIZE or PICTURE.
d. VALUE.

The OCCURS and REDEFINES clauses are not meaningful 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 constant when necessary.

Level 77 items may be mixed with level 01 items in the CONSTANT SECTION.

The clause USAGE IS INDEX must not be used in non-contiguous items.

Constant Records

Constants in the CONSTANT SECTION which bear a definite relationship to one another may be grouped into records according to the rules for formation of record descriptions. With one exception, all record description clauses can be used in a CONSTANT record description, including REDEFINES, OCCURS, and COPY. The clause USAGE IS INDEX cannot be used. Each CONSTANT SECTION record name (01 level) must be unique since it cannot be qualified by a file-name or section-name. Subordinate data-names need not be unique if they can be made unique by qualification.

Value of Constants

The value of every item in the CONSTANT SECTION must be specified by a record description VALUE clause, stated either in the elementary item entry or in the group item entry.

Condition-Names

Since a constant can, by definition, have only one value, there can be no associated condition-names. The use of a condition-name entry (level number 88) in the CONSTANT SECTION is therefore illegal and will constitute a compile error in the source program.

Coding the CONSTANT SECTION

Figure 6-17 illustrates the manner in which the CONSTANT SECTION can be coded.

Burroughs COBOL CODING FORM

PROGRAM: CONSTANT SECTION CODING	REQUESTED BY	PAGE 1 OF 1	
PROGRAMMER: DARIN	DATE 23 SEPT	IDENT. 73	80

```
       CØNSTANT SECTIØN.
       77  TAX-RATE       PIC 9V9(4)   CØMP     VALUE .003.

       01  CITY-CODE-VALUES      SIZE 30.
               05  FILLER  SIZE 15  VA "001005006008011".
               05  FILLER  SIZE 15  VA "011036041059077".
       01  CØDE-REDEFINITIØN REDEFINES CITY-CØDE-VALUES.
               10  CITY-CØDE   ØCCURS 10   PIC 999.

       01  ERRØR-MESSAGE  SZ 24  VA "MASTER FILE CØDE ERRØR.".
```

Figure 6-17. CONSTANT SECTION Coding

LINKAGE SECTION

On the B 7000/B 6000, the LINKAGE SECTION is handled in the same manner as if it were a WORKING-STORAGE SECTION.

DATA-BASE SECTION

The format of the entries in the DATA-BASE SECTION is as follows:

```
01  [internal-set-name]  INVOKE  [LOCAL]  { REFERENCE / REF }  external-set-name

    { IN / OF }  data-base-name.
```

The following example invokes three sets within different data-bases:

```
DATA-BASE SECTION.
01 MAIN-PER INVOKE PERSONNEL-INFOR IN MAIN-CO-BASE.
01 INVOKE INVENTORY IN DB1.
01 REVU INVOKE JOB-REVIEWS IN SECURED-DATA.
```

The DATA-BASE SECTION must be included in the DATA DIVISION if data-management is used in the program.

Briefly, the purpose of the data-management or data-base system is to create and maintain structured data. This data is accessed by a COBOL program using an extended set of verbs. Use of these verbs requires very little programming effort, but gives COBOL programs the ability to INQUIRY, CREATE, and MODIFY data stored in a common data-base. The advantages and use of a data-base with the DMSII system are described in the B 7000/B 6000 DMSII HOST LANGUAGE REFERENCE MANUAL, Form No. 5001498.

Data Management sets can be passed or received as parameters in much the same way as files.

The optional word LOCAL means that the set is local to an external procedure in the DECLARATIVES.

The optional word REFERENCE means that the set is received as a parameter by name. Sets may be received and passed only by REFERENCE.

A data-base is composed of one or more sets. Each set that is opened within the program must be the object of an INVOKE declaration. All items (including other sets) embedded within an invoked set are available to the program, provided the program satisfies the necessary security restrictions to use the set. INVOKE is a declaration of data in the data-base to be used by the program; the internal-set-name is an optional identifier which may be used locally within the COBOL program to refer to the external set-name in the data-base.

LOCAL-STORAGE SECTION

The optional LOCAL-STORAGE SECTION describes parameters received by a procedure when it is invoked.

LD local-storage-name.

The LD entry is followed by item descriptions as used in the WORKING-STORAGE SECTION.

Local storage is associated with a specific procedure by the USE statement mentioning the local-storage-name. The local-storage-name must be unique. An LD entry is required for each procedure that receives data as parameters (i.e., the USING clause is used in both the invocation of the procedure and the USE statement in the section header).

7. 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. The purpose of the following discussion is to explain this division and its elements.

RULES OF PROCEDURE FORMATION

A procedure is composed of a paragraph, a group of successive paragraphs, a section, or a group of successive sections within the PROCEDURE DIVISION. If declaratives are specified, then sections must be used in the remainder of the PROCEDURE DIVISION. A procedure-name is either a paragraph-name or a section-name.

The end of the PROCEDURE DIVISION (the physical end of the program) is that physical position in a COBOL source program after which no further procedures appear.

A section consists of a section header followed by one or more successive paragraphs. A section ends immediately before the next section-name, at the end of the PROCEDURE DIVISION, or in the declaratives portion of the PROCEDURE DIVISION at the key words END DECLARATIVES.

A paragraph consists of a paragraph-name followed by one or more successive sentences. A paragraph ends immediately before the next paragraph-name or section-name or at the end of the PROCEDURE DIVISION.

A sentence consists of one or more statements and is terminated by a period followed by a space.

A statement is a syntactically valid combination of words and symbols beginning with a COBOL verb.

The term "identifier" is defined as the word or words necessary to make unique reference to a data item.

EXECUTION OF PROCEDURE DIVISION

Execution begins with the first statement of the PROCEDURE DIVISION, excluding declaratives. Statements are then executed in the order in which they are presented for compilation, except where the rules in this section indicate some other order.

PROCEDURE DIVISION STRUCTURE

The PROCEDURE DIVISION is identified by, and must begin with, the following header:

```
PROCEDURE DIVISION [ USING { data-name         } ... ] .
                           { file-name         }
                           { control-point-name }
                           { event-name        }
                           { lock-name         }
```

The purpose of the optional USING clause is to name those identifiers which are received as parameters.

Data-names, event-names, control-point-names, and lock-names in the USING clause of the PROCEDURE DIVISION header must have a level-number of 01 or 77.

When the USING clause is present, the object program operates as if each identifier in the list is replaced by the corresponding identifier from the USING clause of the CALL, PROCESS or RUN statement of the calling program or as parameters on a RUN or EXECUTE card. When the RECEIVED BY REFERENCE clause appears in an identifier's data description, the corresponding identifiers refer to a single set of data available to both the calling and called programs. When the identifiers are RECEIVED BY CONTENT, the invocation of the procedure will initialize the corresponding data-name in the called program's USING clause to the current value in the initiating program. The correspondence is positional and not by symbolic name.

Examples:

Program Contains:

```
PROCEDURE DIVISION USING INITIAL-VAL,
   OPTION-VAL, BEAST-NUMBER.
```

Program is executed by control card containing:

```
RUN AND/JUMP (7, "I/OTEST", 666); END
```

The data names in the USING clause will be assigned the values from the control card in the order of their appearance. The COBOL program AND/JUMP must be compiled to run at level 2.

In the above example:

a. INITIAL-VAL and BEAST-NUMBER must be described as 77 level COMP-1 items.

b. OPTION-VAL must be an 01 COMP array declared with LOWER-BOUNDS and BY REFERENCE clauses. The MCP requires this type of a formal parameter description when the corresponding actual parameter is a string. This array may be redefined with an item of USAGE DISPLAY, so that the contents of the array may be more easily accessed.

No item may appear more than once in the same USING clause.

The body of the PROCEDURE DIVISION must conform to the following format:

```
PROCEDURE DIVISION [ USING { data-name
                           { file-name
                           { control-point-name } ... ] .
                           { event-name
                           { lock-name

[MONITOR statement.]
[DUMP statement.] ...
[DECLARATIVES.
 section-name SECTION.  declarative-statement.
 paragraph-name.  [statement.] ...
[paragraph-name.  [statement.] ... ] ...
[section-name SECTION.  declarative-statement.
 paragraph-name.  [statement.]  ...
[paragraph-name.  [statement.] ... ] ... ] ...
 END DECLARATIVES.]
[[section-name SECTION  [priority-number] .]
   paragraph-name.  [statement.] ...
 [[paragraph-name.]  ...  [statement.] ...] ...] ...
```

STATEMENTS

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

Imperative Statements

An imperative statement indicates a specific action to be taken by the object program.

An imperative statement consists of either a verb followed by its operand, or a sequence of imperative statements. Any statement that is not a conditional statement, a compiler-directing statement, or a USE statement is an imperative statement.

Conditional Statements

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

A conditional statement is (1) an IF or SEARCH statement, (2) a READ or RETURN statement that specifies the AT END phrase, (3) a READ or WRITE statement that specifies an INVALID KEY phrase, (4) a WRITE statement that specifies the END-OF-PAGE phrase, (5) a READ, WRITE, AWAIT or WAIT statement that specifies the ON EXCEPTION phrase, (6) a LOCK statement that specifies the AT LOCKED phrase, or (7) the arithmetic statements ADD, SUBTRACT, COMPUTE, DIVIDE, or MULTIPLY that specify the optional phrase ON SIZE ERROR. For example, the IF statement syntax is as follows:

```
IF conditional { statement-1      } [ ELSE { statement-2      } ]
               { NEXT SENTENCE    }       { NEXT SENTENCE    }
```

Statement-1 or statement-2 can be either imperative or conditional statements. If conditional, the statement can, in turn, contain conditional statements in arbitrary depth. Also, if statement-1 or statement-2 is conditional, then the conditions within the conditional statement are considered to be "nested."

Compiler-Directing Statements

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

SENTENCES

A sentence consists of one or more statements, the last of which is terminated by a period. A sentence composed of a compiler-directing statement is called a compiler-directing sentence. A sentence composed of imperative and/or conditional statements is called a procedural sentence.

Imperative Sentences

One or more imperative statements terminated by a period is an imperative sentence.

Examples:

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

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

Example:

```
MOVE A TO B; ADD C TO D; GO TO START.
```

Conditional Sentences

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

Example:

```
IF X EQUALS Y
    MOVE A TO B
    IF W EQUALS T
        ADD A TO B
    ELSE NEXT SENTENCE
ELSE MOVE C TO D.

MOVE Y TO A
OPEN OUTPUT ERROR-FILE
ADD X TO Y ON SIZE ERROR ADD Y TO Z DISPLAY "OVERFLOW ON Y".

READ FILE-A
    AT END GO TO CLOSING-ROUTINE ELSE
MOVE P TO R.
```

If the phrase ELSE NEXT SENTENCE immediately precedes the period, then this phrase may be eliminated. This rule may then be applied again to the resulting sentence.

Compiler-Directing Sentences

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

SENTENCE PUNCTUATION

The following rules apply to the punctuation of sentences:

a. A sentence is terminated by a period followed by a space.
b. A separator is a word or character used for the purpose of enhancing readability. Use of a separator is optional.
c. The allowable separators are the semicolon (;) and the comma (,).
d. Separators must not be followed immediately by another separator.
e. Separators may be used only 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.

SENTENCE EXECUTION

For the remainder of this discussion, by "execution of a sentence or a statement within a sentence" is meant "execution of an object program compiled from a sentence, or from a statement within a sentence which has been written in COBOL. By "transfer of control" is meant "transfer of control in the object program by transferring (GO ing) from one place to another out of the written sequence." By "passing of control" is meant "passing of control in the object program by passing from one place to the next place in the written sequence."

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

Imperative Sentences

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

Conditional Sentences

A condition causes the object program to select between alternate paths of control depending upon the truth value of a test.

```
IF condition {statement-1      } [ELSE {statement-2      }]
             {NEXT SENTENCE    }       {NEXT SENTENCE    }
```

The condition is an expression which is TRUE or FALSE. If the condition is TRUE, statement-1 is executed and control is then implicitly transferred to the next sentence unless statement-1 causes some other transfer of control. If the condition is FALSE, statement-2 is executed and control is then passed to the next sentence unless statement-2 causes some other transfer of control.

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
    ELSE statement-4
ELSE statement-2
```

If condition-1 is TRUE, imperative-statement-1 is executed; then if condition-2 is TRUE, statement-3 is executed and control is transferred to the next sentence. If condition-2 is FALSE, then statement-4 is executed and control is transferred to the next sentence. If condition-1 is FALSE, statement-2 is executed and control is passed to the next sentence. Statement-3 can in turn be either imperative or conditional and, if conditional, can in turn contain conditional statements in arbitrary depth. In an identical manner, statement-4 can either be imperative or conditional.

The execution of the phrase NEXT SENTENCE causes a transfer of control to the next sentence as written, except when it appears in the last sentence of a procedure being PERFORMED, in which case control is passed to the return mechanism.

USASI-1968 and ANSI-1974 specify that the statements associated with the special conditions AT END, INVALID KEY, END-OF-PAGE, and ON SIZE ERROR must be imperative statements. B 7000/B 6000 COBOL does not have this restriction. These special conditions (which imply a test) are handled the same as a condition associated with an IF verb; that is, they may be logically paired with an ELSE, and statements associated with them may be either imperative or conditional statements. The system options USASI, ANSI74, or S360 can be used to cause ELSE to be matched only to IF statements.

Consider:

```
IF condition-1

        READ file-name
        AT END
           IF condition-2 statement-1

           ELSE statement-2
        ELSE statement-3
   ELSE statement-4
```

If condition-1 is TRUE, the READ statement is executed. If the logical end of the file is detected (the AT END is TRUE), condition-2 is tested. If condition-2 is TRUE, statement-1 is executed and control is transferred to the next sentence. If condition-2 is FALSE, statement-2 is executed and control is transferred to the next sentence. If the AT END condition is FALSE, statement 3 is executed and control is transferred to the next sentence. Note that the ELSE immediately preceding statement-3 is considered to be logically paired with the AT END condition. Statement-4 is executed only when condition -1 is FALSE. With the system options USASI, ANSI74 or S360 set, an ELSE will be paired only with an IF statement and never with an AT END, INVALID KEY, ON SIZE ERROR, AT END-OF-PAGE or ON EXCEPTION clause. In the above example, the statement-3 would be executed if condition-1 were false. The ELSE statement-4 would produce a syntax error.

For a further discussion of conditions, refer to the IF statement.

Compiler-Directing Sentences

Compiler-directing sentences direct a COBOL processor to take action at compilation time. On the other hand, procedural sentences denote action to be taken by the object program. Compiler-directing sentences result in inclusion of routines in the object program and do not directly result in either the transfer or passing of control. The routines themselves, which the compiler-directing sentences 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 to be accomplished. The sentences are written successively, according to the rules of the coding form, 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 control to a named procedure.

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

PARAGRAPHS

In order that the 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, the programmer must begin a paragraph with a name. The name consists of a noun followed by a period, and the name precedes the paragraph it names. A paragraph is terminated by the next paragraph or section-name. The smallest named grouping of the PROCEDURE DIVISION is a paragraph.

SECTIONS

A section consists of one or more successive paragraphs. 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) may begin on the same line as the section header. 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 preferable, but not required, that a program be subdivided into sections, each section containing a functional part of the program.

DECLARATIVES

General Description

Declaratives are procedures which operate under the control of the "main body" of the PROCEDURE DIVISION or by means of the input-output system. Declaratives, if present, must be grouped together at the beginning of the PROCEDURE DIVISION. The group of DECLARATIVES must be preceded by the key word DECLARATIVES, and followed by the words END DECLARATIVES. Each declarative consists of a single section and must conform to the rules for procedure formation.

USE Declarative

There are three major types of USE declaratives. The first type allows additional procedures to be provided for input and output errors and label handling. The second is more general, allowing the user to create logical subdivisions within a program. A third type is the interrupt procedure.

More information on the USE declaratives may be found in the discussion of the USE statement.

FORMULAS

A formula is an algebraic expression consisting of a combination of arithmetic expressions and intrinsic functions. An arithmetic expression can be any of the following: an identifier of a numeric item, a numeric literal, identifiers and literals separated by arithmetic operators, two arithmetic expressions separated by an arithmetic operator, or an arithmetic expression enclosed in parentheses. Negative values can be expressed by using a unary minus. The permissible combinations of identifiers, literals, and arithmetic operators are given in figure 7-2.

Those identifiers and literals appearing in an arithmetic expression must represent either numeric elementary items or numeric literals on which arithmetic operations may be performed.

The specific types of operands that can appear in arithmetic formulas are as follows:

1. Numeric data items.
2. Numeric literals.
3. Intrinsic functions (including algebraic typed installation intrinsics). Boolean intrinsics are treated as Boolean when used in a conditional expression, and as integer functions when used in arithmetic-expressions.
4. The figurative constant ZERO, ZEROES, ZEROS.
5. Special registers and attributes whose class is implicitly numeric; included in this category are:
   A. The TIME and COMPILETIME functions.
   B. TODAYS-DATE.
   C. TALLY.
   D. CHECKPOINT-STATUS.
   E. LINAGE-COUNTER.
   F. The attribute mnemonic value functions.
   G. Any file, task, or direct I/O area attribute whose implicit class is numeric.
   H. The various Data Management status functions.

FORMULAS

Basic Operators

There are five basic symbols available to express binary and unary arithmetic operations. These symbols, and their English equivalents, are shown in figure 7-1.

CHARACTER	ENGLISH EQUIVALENT
+	PLUS
-	MINUS
*	MULTIPLIED BY
/	DIVIDED BY
**	EXPONENTIATED BY

Figure 7-1. Arithmetic Operators

The symbol - must be preceded and followed by a space. The exponentiation symbol ** cannot contain an embedded space; however, the symbol may be split across source cards.

A plus sign or minus sign immediately preceding a numeric literal (with no intervening spaces) becomes a part of that literal, making it a signed numeric literal. In this case, the sign is neither a binary nor unary operator. For example, A +2 is equivalent to A, +2. The latter is two separate expressions, as would appear in a subscript list. A plus sign in any other situation is treated as a binary operator if preceded by an operand and as a unary operator if not preceded by an operand. For example, A + 2 and A + +2 are equivalent expressions.

The rules for forming algebraic expressions assume the existence of a precedence table for operations which, unless parenthesizing is used to modify hierarchy, determines the sequence in which the operations in a formula will be executed. Normal precedence, from high to low, is as follows:

a. Unary operations (including intrinsic functions).
b. Exponentiation.
c. Multiplication and division.
d. Addition and subtraction.

The symbols + and - are also used to indicate unary options. If these symbols are used without parenthesizing, they 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 deviate 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, for example, 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 the use of parentheses the following example illustrates normal precedence:

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

and would be interpreted as:

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

with the sequence of operations proceeding from the innermost parentheses toward the outside; i.e., first exponentiation, then multiplication and division, and finally addition and subtraction.

Two additional basic operators, MOD and DIV, have been implemented for use in arithmetic operations. They are syntactically expressed as follows:

dividend $\left\{ \begin{matrix} \underline{\text{MOD}} \\ \underline{\text{DIV}} \end{matrix} \right\}$ divisor

The operators MOD and DIV must be preceded and followed by a space. They correspond in precedence to multiplication and division. MOD is commonly called a "remainder divide." The result of this operation is the remainder of the division.

Examples:

```
(17 MOD 9) = 8
(10 MOD 8) = 2
(156 MOD 120) = 36
```

DIV is commonly called an "integer divide." The result of this operation is the integer part of the quotient after division.

Examples:

```
(10 DIV 8) = 1
(10 DIV .33) = 30
(20 DIV 11) = 1
```

Formation of Symbol Pairs

The ways in which symbol pairs may be formed are summarized in figure 7-2 below, where:

a. The letter P represents a permissible pair of symbols.

b. The character - represents an invalid pair.

c. Variable represents an identifier or literal.

FIRST SYMBOL \ SECOND SYMBOL	VARIABLE	** / * + -	UNARY + OR -	(	)
VARIABLE	-	P	-	-	P
** / * + -	P	-	P	P	-
UNARY + OR -	P	-	-	P	-
(	P	-	P	P	-
)	-	P	-	-	P

Figure 7-2. Formation of Symbol Pairs in Arithmetic Expressions

An arithmetic expression may only begin with the symbol for a left parenthesis, for a minus or plus sign or with a variable, and may only end with a right parenthesis or a variable. There must be a one-to-one correspondence between left and right parentheses of an arithmetic expression such that each left parenthesis is to the left of its corresponding right parenthesis.

INTRINSIC FUNCTIONS

The B 7000/B 6000 compiler provides a set of intrinsic functions which may be included in formulas. The format for calling these functions in a formula is:

function-name (formula)

(The parentheses are required.) Figure 7-3 lists the function-names and their functions:

FUNCTION-NAMES	FUNCTIONS
SIN	Calculates the Sine.
COS	Calculates the Cosine.
ARCTAN	Calculates the Arctangent.
EXP	Calculates the Exponential Function; that is, e ** Formula.
SIGN	Returns +1, -1, or 0, depending on whether the formula is greater than, less than, or equal to 0.
SQRT	Calculates the square root.
LN	Calculates the natural log (to the base e).
ABS	Calculates the absolute value.
ONES	Returns the number of non-zero bits in the parameter value. The parameter may be either a single or double precision expression.
FIRSTONE	Returns the bit number, plus one, of the leftmost non-zero bit in the parameter value. If the parameter value is double precision, only the first (higher order) word is used. If the parameter value is zero, FIRSTONE returns zero.
MAX	Returns the maximum of the parameter values.
MIN	Returns the minimum of the parameter values.

Figure 7-3. Arithmetic Intrinsic Functions

SIN, COS, ARCTAN functions assume the angle to be in radians. Double-precision accuracy is obtained if the argument is double precision. A mathematical explanation of these functions and of the method used for their calculation is contained in the B 7000/B 6000 System Software Operational Guide, Volume 1, Form No. 5001563.

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, PERFORM, and SEARCH statements.

A condition is one of the following:

a. Relation Condition.
b. Sign Condition.
c. Class Condition.
d. Condition-Name Condition.
e. Event-Identifier Condition.
f. Not Condition.
g. Condition (AND/OR) Condition [(AND/OR) Condition]... .

Any condition may be enclosed in parentheses. The truth value of a parenthesized condition is determined from the evaluation of the truth values of its constituents. A parenthesized condition is a condition in the sense of the last two items of the preceding list.

The construction,

NOT condition

where condition is one of the conditions listed above, is not permitted if the condition itself contains a logical NOT. The logical operator NOT indicates the negation of the condition that follows it.

Conditions may be combined by means of logical operators. The meaning of the logical operators is as follows:

LOGICAL OPERATOR	MEANING
OR	Logical Inclusive OR
AND	Logical Conjunction
NOT	Logical Negation

Figure 7-4 indicates the relationships between the logical operators and conditions A and B. Figure 7-5 indicates the ways in which conditions and logical operators may be combined.

CONDITION		RESULT FOR LOGICAL TEST		
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

Figure 7-4. Relationship of Conditions, Logical Operators, and Truth Values

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

Figure 7-5. Combinations of Conditions and Logical Operators

NOTE

"P" indicates that the pair is permissible, and "-" indicates the pair is not permissible. Thus the pair "OR NOT" is permissible, but the pair "NOT OR" is not permissible.

*Permissible only if the condition is not itself a "NOT condition".

Relation Condition

A relation condition causes a comparison of two operands, each of which may be an identifier, literal, or arithmetic expression.

The general format for a relation condition is as follows:

```
{identifier-1           }                          {identifier-2           }
{literal-1              } relational-operator      {literal-2              }
{arithmetic-expression-1}                          {arithmetic-expression-2}
```

The first operand (identifier-1, literal-1, or arithmetic-expression-1) is called the subject of the condition, the second operand (identifier-2, literal-2, or arithmetic-expression-2) is called the object of the condition.

The relational operators specify the type of comparison to be made in a relation condition.

The following are the relational operators:

```
IS [NOT] GREATER THAN
IS [NOT]  >
IS [NOT] LESS THAN
IS [NOT]  <
IS [NOT] EQUAL TO
IS [NOT]  =
IS UNEQUAL TO
EQUALS
EXCEEDS
```

Note that the required relational operators ">", "<", and "=" are not underlined, to avoid confusion with other symbols.

COMPARISON OF NUMERIC OPERANDS

For operands whose class is numeric, a comparison is made 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. Group items declared as USAGE COMP may not appear as an operand in a relation condition. Unsigned numeric operands are considered positive for purposes of comparison.

COMPARISON OF NON-NUMERIC OPERANDS

For non-numeric operands, or one numeric and one non-numeric operand, character comparisons occur as shown in table 7-1.

Table 7-1. Comparison of Non-Numeric Operands

Possible Comparisons:		Compared As:
1.	EBCDIC Vs. EBCDIC	EBCDIC, SPACE fill
2.	EBCDIC Vs. BCL	EBCDIC, SPACE fill
3.	EBCDIC Vs. ASCII	EBCDIC, SPACE fill
4.	EBCDIC Vs. HEX	HEX, ZERO fill
5.	BCL Vs. BCL	* EBCDIC, SPACE fill
6.	BCL Vs. ASCII	ASCII, SPACE fill
7.	BCL Vs. HEX	HEX, ZERO fill
8.	ASCII Vs. ASCII	ASCII, SPACE fill
9.	ASCII Vs. HEX	HEX, ZERO fill

NOTE

* BCL Vs. BCL are not translated on EQUAL or NOT EQUAL conditions.

Operands are translated if necessary to EBCDIC before comparison. When both operands are ASCII or COMP-2, no translation takes place. When both operands have the same USAGE and the relational operator is (NOT) EQUAL, (NOT) = , EQUALS or UNEQUAL, no translation takes place. For all other comparisons, operands with a USAGE other than DISPLAY are translated to DISPLAY.

There are two cases of non-numeric comparison to consider: operands of equal size and operands of unequal size.

OPERANDS OF EQUAL SIZE

If operands are of equal size, characters in corresponding character positions of the two operands are compared starting from the high-order end thru the low-order end.

If all pairs of characters compare equally thru the last pair, the operands are considered equal.

The first pair of unequal characters to be encountered is compared to determine their relative position in the collating sequence. The operand that contains the character positioned higher in the collating sequence is considered to be the greater operand.

OPERANDS OF UNEQUAL SIZE

If the operands are of unequal size, the comparison of characters proceeds as if the shorter operand had been expanded by blanks on the right to make it equal in size to the longer operand.

Elementary COMP or COMP-1 items cannot be compared to any of the "non-numeric" figurative constants (i.e., SPACES, QUOTES, HIGH-VALUES, and LOW-VALUES) in a relational condition.

COMPARISONS INVOLVING INDEX-NAMES AND/OR INDEX DATA ITEMS

The result of the comparison of the two index-names is the same as if the corresponding occurrence numbers are compared.

In the comparison of an index-name and a data item (other than an index data item) or literal, the occurrence number that corresponds to the value of the index-name is compared to the data item or literal.

In the comparison of an index data item and an index-name or another index data item, the actual values are compared without conversion.

Comparison of an index data item with any data item or literal not specified above is illegal.

Sign Condition

The sign condition determines whether or not the algebraic value of a numeric operand is less than, greater than, or equal to 0. The general format for a sign condition is as follows:

arithmetic-expression IS [NOT] { POSITIVE / NEGATIVE / ZERO }

An operand is positive if its value is greater than zero, negative if its value is less than zero, and zero if its value is equal to zero.

Class Condition

The class condition determines whether the operand is numeric; that is, consists entirely of the characters 0, 1, 2, 3, ..., 9, with or without an operational sign, or alphabetic, that is, consists entirely of the characters A, B, C, ..., Z, and space. The general format for the class condition is as follows:

identifier IS [NOT] { NUMERIC / ALPHABETIC }

The usage of the operand being tested must be described, implicitly or explicitly, as DISPLAY, DISPLAY-1, ASCII or COMP-2.

The NUMERIC test cannot be used with an item whose data description describes the item as alphabetic. If the record description of the item being tested does not contain an operational sign, the item being tested is determined to be numeric only if the contents are numeric and an operational sign is not present. If the item being tested is described as a signed item, the item being tested is determined to be numeric only if the contents are numeric and the sign is a hex "C" or a hex "D" or a hex "F".

Computation operands cannot be checked with the numeric class condition since they always contain numeric data, regardless of their bit pattern.

The ALPHABETIC test cannot be used with an item whose record description describes the item as numeric. The item being tested is determined to be alphabetic only if the contents consist of any combination of the alphabetic characters A thru Z and the space.

Operands used in the alphabetic class condition must be either DISPLAY, DISPLAY-1, or ASCII. COMP-2 is not allowed with the ALPHABETIC test.

Condition-Name Condition

In a condition-name condition, a conditional variable is tested to determine whether or not its value is equal to one of the values associated with a condition-name. The general format for the condition-name condition is as follows:

(NOT) condition-name

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

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

The result of the test is TRUE if one of the values corresponding to the condition-name equals the value of its associated conditional variable.

Event-Identifier Condition

In an event-identifier condition, an event-identifier is tested to determine whether or not the condition has happened. The general format for the event-identifier condition is as follows:

```
[NOT] event-identifier
```

The result of the test is TRUE if the event has been explicitly caused by the CAUSE statement or implicitly caused by a DIRECT I-O function to which the event is associated; in either case, the event may be caused to happen in the body of the program where it is defined or in the body of any program that is functioning under the control of the program where it is defined.

Evaluation Rules for Conditions

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

a. ARITHMETIC EXPRESSION
b. ALL RELATIONAL OPERATORS
c. NOT
d. AND
e. OR

Abbreviations

When relation conditions are written in a consecutive sequence, any relation except the first may be abbreviated by:

a. The omission of the subject of the relation (implied subject).
b. The omission of the subject and relational operator.
c. The omission of the subject, relational operator, and all but the last logical connector of the relation.

Example of a series of equivalent relational conditions:

```
No Abbreviation      A > B OR A > C OR A  > D
Abbreviation a.      A > B OR   > C OR    > D
Abbreviation b.      A > B OR     C OR      D
Abbreviation c.      A > B        C OR      D
```

Since NOT can be used as both a logical operator and as part of a relational operator, its use in an abbreviated relation condition is interpreted as follows:

1. If no relational operator is explicitly present in the abbreviated relation condition, the NOT is always interpreted as a logical operator.

   For example:

   A > B OR NOT C OR D

   Is compiled as:

   A > B OR NOT (A > C) OR A > D

2. If the NOT is preceded by the word IS and is followed by one of the words GREATER, LESS, EQUAL, or by one of the symbols > ,< , or =, then the NOT is interpreted as being a part of an explicit relational operator.

   For example:

   A > B OR IS NOT > C OR D

   Is compiled as:

   A > B OR A IS NOT > C OR A IS NOT > D

3. If the NOT is followed by either of the words GREATER, LESS, EQUAL, or one of the symbols > , < , or =, but is not preceded by the optional word IS, then the NOT is interpreted as a logical operator; however, if the ANSI74 system option is set, the NOT is interpreted as part of the relational operator. This difference becomes important if there are subsequent abbreviated relation conditions, because logical operator NOTs do not apply to subsequent relation conditions.

   For example:

   A > B AND NOT < C OR D

   Is compiled as:

   A > B AND NOT (A< C) OR A< D (when ANSI74 is RESET)

   But is compiled as:

   A > B AND A NOT < C OR A NOT < D (when ANSI74 is SET)

CONDITIONS

The following rules govern implied subjects, relational operators, and logical connectors:

a. The implied subject is always the last stated subject, regardless of parentheses.
b. The implied relational operator is always the last stated relational operator, regardless of parentheses.
c. The implied logical connector is always the last stated logical connector, regardless of parentheses.

For example:

A > B OR C OR (D = F OR G OR H) OR J

The implied subject and relational operator for C are A and > , respectively.
The implied subject and relational operator for J are D and =, respectively.

STATEMENT OPTIONS

In the statement descriptions that follow, several options appear frequently: the ROUNDED option, the SIZE ERROR option, and the CORRESPONDING option.

In the discussion below, a resultant-identifier is that identifier associated with a result of an arithmetic operation.

Rounded Option

If after decimal point alignment, the number of places in the fraction of the result of an arithmetic operation is greater than the number of places provided for the fraction of the resultant identifier, truncation is relative to the size provided for the resultant-identifier. When rounding is requested, the value returned is (X + 0.5), where X is the original argument.

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

Size Error Option

If, after decimal point alignment, the value of a result exceeds the largest value that can be contained in the associated resultant-identifier, a size error condition exists. The size error condition applies only to the final results of an arithmetic operation and does not apply to intermediate results, except in the MULTIPLY and DIVIDE statements, in which case the size error condition applies to the intermediate results as well. Arithmetic faults such as divide by zero and integer overflow will cause program termination if the SIZE ERROR clause is not used.

If the ROUNDED option is specified, rounding takes place before checking for size error. When such a size error condition occurs, the subsequent action depends on whether or not the SIZE ERROR option is specified:

a. If the SIZE ERROR option is not specified and a size error condition occurs, the values of the resultant-identifier(s) affected will be unpredictable. Values of resultant identifier(s) for which no size error condition occurs are unaffected by size errors that occur for other resultant-identifier(s) during execution of this operation.

b. If the SIZE ERROR option is specified and a size error condition occurs, then the values of resultant-identifier(s) affected by the size errors are not altered. Values of resultant-identifier(s) for which no size error condition occurs are unaffected by size errors that occur for other resultant-identifier(s) during execution of

this operation. After completion of the execution of this operation, the statement in the SIZE ERROR option is executed.

For ADD and SUBTRACT CORRESPONDING, the statement in the SIZE ERROR clause is not executed until all of the individual additions or subtractions are completed.

c. For data items described as COMP or COMP-1, only the physical size is considered for size error conditions. This may be overridden for COMP items by setting either of the $ options USASI or ANSI74 to then cause size error testing to be based on the declared decimal size rather than the physical binary size.

Corresponding Option

In this discussion, identifier-1, identifier-2, identifier-3,..., are identifiers specified in a statement containing the CORRESPONDING phrase.

a. Rules for valid identifiers are:

   1. All identifiers must refer to group items.
   2. Identifiers may be described with or be subordinate to an item described with a REDEFINES or OCCURS clause.
   3. No identifier may have a USAGE of INDEX, EVENT, LOCK or CP.

b. Data items subordinate to identifier-1 correspond with data items subordinate to identifier-2, identifier-3,..., if the following rules apply:

   1. Both data items must have the same data-name.
   2. All possible qualifiers for the sending item, up to but not including identifier-1, must be identical to all possible qualifiers for the receiving item up to but not including identifier-2, identifier-3,... .
   3. In an ADD or SUBTRACT statement, only elementary numeric data items will be considered.
   4. In a MOVE statement, the corresponding sending and/or receiving data items must be elementary. The class may differ.
   5. Any item with a level-number of 66 or 88 or with a Data Description entry containing a REDEFINES, OCCURS, INDEX, EVENT, LOCK or CP clause is not considered. Any item subordinate to an item not eligible for correspondence will also be ignored.
   6. FILLER data items are ignored.

VERBS

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

ARITHMETIC	INPUT/OUTPUT	PROCEDURE BRANCHING	
ADD	ACCEPT	ALTER	EXIT
COMPUTE	CLOSE	CALL	GO
DIVIDE	DISPLAY	CONTINUE*	PERFORM
MULTIPLY	OPEN	ENTER	PROCESS
SUBTRACT	READ		RUN
	SEEK		
	WRITE		

DATA MOVEMENT	ENDING	COMPILER-DIRECTING
EXAMINE	STOP	COPY
MOVE		DUMP *
INSPECT		MONITOR *
STRING		
UNSTRING		

SORT VERBS	MCP, PROGRAM COMMUNICATION, TASKING		
MERGE	ALLOW *		LOCK *
RELEASE	ATTACH *	DEALLOCATE	RESET *
RETURN	AWAIT *	DETACH *	UNLOCK *
SORT	CAUSE *	DISALLOW *	USE
	CHECKPOINT	EXECUTE *	WAIT *

TABLE MANIPULATION	CONDITIONAL
SEARCH	IF
SET	

* B 7000/B 6000 Extensions.

Although the word IF is not a verb in the English language, it is a verb in the COBOL language because it possesses the characteristics of generating code in the object program. Its occurrence is a vital feature in the PROCEDURE DIVISION.

ACCEPT

The function of the ACCEPT statement is to permit the entry of low-volume data from the console keyboard.

The ACCEPT statement format consists of two options as follows:

Option 1:

```
ACCEPT identifier [FROM {hardware-name}]
                       {mnemonic-name}
```

The ACCEPT statement generates a message on the console which indicates the data-name to be entered. The program is then suspended until the operator enters the data. The identifier may be any elementary or group item other than an index-data-item, index-name, EVENT, LOCK or CP.

Group items must have DISPLAY or DISPLAY-1 usage, and no more than 256 characters may be accepted at one time.

Because of the slow speed involved in entering information thru the keyboard, the ACCEPT statement should be used sparingly and solely for low-volume data entry.

The FROM option is optional and is used for documentation only. When specified, hardware-name must be MESSAGE-PRINTER, SPO, KEYBOARD or DISPLAY-UNIT. Mnemonic-name must be associated with one of the above hardware-names in SPECIAL-NAMES.

Data entered by the operator will be left justified in identifier. Excess characters on the right hand side will be truncated. Operator response to an ACCEPT will consist of the mix number of the task preceeding or following the letters "AX", then an optional space or spaces and then the data to be entered.

Option 2:

```
ACCEPT identifier FROM {DATE}
                       {DAY }
                       {TIME}
```

Option 2 is an ANSI 74 implementation which requires that the ANSI74 system dollar option be set.

DATE is comprised of the data elements of year, month, and day in the sequence left to right. Therefore, July 1, 1968 would be expressed as 680701. DATE, when accessed by a COBOL program, behaves as if it has been described as an unsigned, elementary numeric integer data item six digits in length.

DAY is comprised of the data elements of year and day of year, left to right. Therefore, July 1, 1968, would be expressed as 68183. DAY is treated as an unsigned elementary numeric integer data item five digits in length.

TIME is composed of the data elements hours, minutes, seconds and hundredths of a second. TIME starts at midnight on a 24-hour clock basis. Therefore, 2:41 PM would be expressed as **14410000.** TIME is treated as an unsigned numeric integer data item eight digits in length.

An ACCEPT statement implies a MOVE of day, date or time to the identifier by normal MOVE rules. Refer to Appendix B for a description of the ANSI 74 implementations.

ADD

The ADD statement causes two or more numeric operands to be summed and the result to be stored.

The format of the ADD statement consists of three options as follows:

Option 1:

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

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

     [; ON SIZE ERROR statement [ELSE statement]]
```

Option 2:

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

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

     [; ON SIZE ERROR statement [ELSE statement]]
```

Option 3:

```
ADD  {CORRESPONDING} identifier-1 TO identifier-2 [ROUNDED]
     {CORR         }

     [; ON SIZE ERROR statement [ELSE statement]]
```

In options 1 and 2, each identifier must refer to an elementary numeric item, except that identifiers appearing only to the right of the word GIVING may refer to data items that contain editing symbols.

Each literal must be a numeric literal.

The maximum size of any operand, literal, or identifier is 23 decimal digits. The composite of operands, which is that data item resulting from the superimposition of all operands (excluding the data items that follow the word GIVING), aligned on their decimal points, must not contain more than 23 digits.

If option 1 is used, the value of all operands preceding the word TO are added together; then the sum is added to the current value of each identifier-m, identifier-n,....

If option 2 is used, the values of the operands preceding the word GIVING are added together, and the sum is stored as the new value of each identifier-m, identifier-n.

If option 3 is used, data items in the group item referred to by identifier-1 are added to and stored in the corresponding data items of the identifier-2 group item.

The internal format of operands referred to in an ADD statement may differ. Any necessary format transformations and decimal point alignment are automatically supplied throughout the execution.

Either statement shown in the ON SIZE ERROR clause may be conditional or imperative, or the reserved words NEXT SENTENCE may be used instead.

ALLOW

The ALLOW statement permits the execution of an interrupt procedure that has been attached to an EVENT item. The format of the ALLOW statement is as follows:

```
ALLOW  {section-name-1 [,section-name-2]...}
       {INTERRUPT                           }
```

Any section-names used in this statement must be defined as interrupt procedures with the USE statement (option 4). These section-names must be attached to EVENT items at the time of execution of this statement by the prior execution of an ATTACH statement.

ALLOW INTERRUPT causes all interrupt procedures which are attached to event-identifiers to be allowed.

The interrupt procedure will not be executed until a CAUSE statement has been executed for the EVENT named in the ATTACH statement and the ALLOW statement has been executed for the section-name associated (via the ATTACH statement) with the EVENT.

ALTER

The ALTER statement modifies a predetermined sequence of operations by changing the operand of a labeled GO statement.

The format for the ALTER statement is as follows:

```
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. Procedure-name-2, procedure-name-4, ... are either paragraph or section names, or paragraph names qualified by section names.

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

ATTACH

The ATTACH statement establishes an association of an interrupt procedure to an EVENT item.

The format of the ATTACH statement is as follows:

ATTACH section-name TO event-identifier

The interrupt procedure must not already be attached to some other EVENT at the time the ATTACH statement is executed. Section-name must be the name of a section in the Declaratives which specifies an option 4 USE.

The causation of an EVENT item to which an interrupt procedure is attached results in the queueing of all interrupt procedures which are then attached to that event-identifier. They remain queued until such time as each procedure is implicitly or explicitly referenced by an ALLOW statement.

When an interrupt procedure is attached to an event-identifier, the procedure is automatically allowed. The DISALLOW statement will inhibit that interrupt procedure so that subsequent executions of an appropriate CAUSE statement will not bring about execution of the interrupt procedure.

Two or more interrupt procedures may be attached to a single event-identifier. The order of execution of these interrupt procedures will be in descending order of attachments.

AWAIT (WAIT)

The AWAIT statement suspends the execution of the procedure or task in which it appears.

The format and use of the AWAIT statement are described in this section, under the heading WAIT. AWAIT is synonomous with the WAIT statement.

CALL

The CALL statement causes control to be transferred from one procedure to another. The function of this statement is similar to, but distinct from, the function of the PERFORM statement.

The format of the CALL statement is as follows:

Option 1:

```
CALL control-point-identifier WITH section-name
  [USING actual-parameter-list]
```

Option 2:

```
CALL { section-name           } [USING actual-parameter-list]
     { installation-intrinsic }
```

Option 3:

```
CALL PROGRAM DUMP
```

Option 4:

```
{ CALL SYSTEM WITH } { data-name }
{      ZIP        } { file-name }
```

Option 1 is used to create a co-routine. Option 2 is used to call as a procedure an externally compiled procedure that will be bound to the calling program. (Option 2 is synonymous with the ENTER statement.) Untyped installation intrinsics can be called via Option 2 of the CALL statement. Actually, either typed or untyped installation intrinsics can be called. However, the value returned by typed intrinsics will be deleted. Option 3 causes a PROGRAMDUMP. Option 4 is used to pass a control message to the MCP.

Each identifier in the USING clause must be defined as either a 77 level item that resides in the stack or an 01 level and must correspond as to level number, usage, and size of the items described in the corresponding positions of the CALLed program's PROCEDURE DIVISION USING clause. The identifiers in the USING clause may be any combination of control-point items, data items, EVENT items, INDEX or LOCK items at either the group or elementary level.

The USING clause is included in the CALL statement if, and only if, there is a USING clause in the USE statement of the section name or in the PROCEDURE DIVISION header of the called program. The number, type, and order of items in each USING clause must be identical.

The execution of the CALL statement causes the program containing the CALL to be suspended and the program or intrinsic being called to be put into execution. Upon execution of the called program/s EXIT PROGRAM statement, the called program is suspended and control is returned to the calling program's next statemen

The inclusion of a control-point item in the USING clause allows a called program to make any reference to the control-point that is allowable in the calling program.

Files to be passed as parameters must have a record description unless the file is a DIRECT file. The record described for the file may be passed as a parameter. In the PROCEDURE DIVISION header of the CALLed program, the USING phrase must not reference any data item in the CALLed programs FILE SECTION. Either or both programs may initiate I/O to the file passed as file-name-1 in the CALL statement.

If event-name-1 is included in the USING list, it must be a 77 or 01 level item declared with USAGE IS EVENT and may be caused in either or both the called and calling programs.

Figure 7-6 shows that the name of a program to be called can be either specified at the source level or can be set at object time by moving or reading the code file title into the data-name associated with S2. The contents of data-name must be one to 14 names, separated by slashes (/), with a period (.) immediately following the last (or only) name. Each of the names may be a maximum of 17 characters.

In figure 7-6, the called program must contain a USING clause in its PROCEDURE DIVISION header whose data-names correspond as to level number, size, and usage with R and X. See the RECEIVED clause for further examples.

Attempts to alter the contents of NAME-TO-BE-CALLED after the first call of B, but prior to a DETACH, will be ignored.

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  B-6700.
OBJECT-COMPUTER.  B6700.
SPECIAL-NAMES.  "A"/"COMPILED"/"CODEFILE" IS TO-BE-CALLED.
INPUT-OUTPUT SECTION.
FILE-CONTROL.  SELECT LOCAL REF FILE-PARAM ASSIGN TO DISK.
               SELECT HISTORY ASSIGN TO DISK.
DATA DIVISION.
FILE SECTION.
FD FILE-PARAM; RECORD CONTAINS 132 DISPLAY.
01 PARAM-RECORD SIZE 132.
FD HISTORY; RECORD CONTAINS 132 DISPLAY.
01 HIST-RECORD SIZE 132.
WORKING-STORAGE SECTION.
77 RR PIC 9(6) COMP-1.
77 XX PIC 9(6) COMP-1.
01 NAME-TO-BE-CALLED PIC X(20) DISPLAY.
01 CPA USAGE IS CONTROL-POINT.
01 CPB USAGE IS CONTROL-POINT.
LOCAL-STORAGE SECTION.
LD LDX.
77 R REF PIC 9(6) COMP-1.
77 X REF PIC 9(6) COMP-1.
PROCEDURE DIVISION.
DECLARATIVES.
S1 SECTION.  USE EXTERNAL TO-BE-CALLED AS PROCEDURE
             WITH LDX, FILE-PARAM USING R,X,FILE-PARAM.
S2 SECTION.  USE EXTERNAL NAME-TO-BE-CALLED AS PROCEDURE
             WITH LDX USING R,X.
END DECLARATIVES.
MAIN SECTION.
CALLING.  CALL CPA WITH S1 USING RR,XX,HISTORY.
          IF XX = 3 MOVE "TIO/PROCTOR."  TO NAME-TO-BE-CALLED.
    ELSE ACCEPT NAME-TO-BE-CALLED.  CALL CPB WITH S2 USING XX,RR.
EOJ.  STOP RUN.
```

Figure 7-6. Example of Calling Another Program

When Option 4 is executed, the MCP interprets the control message and performs the operation(s) specified. Figure 7-7 is an example of an Option 4 CALL statement.

NOTE: ZIP may be used as a synonym for CALL SYSTEM WITH, but only while the B2500 system dollar option is set. This feature provides compatibility with B3700 COBOL.

If the data-name option is used, the data-name must be the name of a level 01 data item whose usage is DISPLAY or DISPLAY-1. The first character of data-name must not be used because the compiler inserts an invalid character in this location. The content of data-name must contain the entire control message in standard card format, (i.e., EXECUTE A/B; FILE Q(TITLE = X/H); END).

If the file-name option is used, the file-name must be the name of a file in a format consistent with that used for a load-control "PSEUDO-DECK". Briefly this is a card-image file which has a fixed length (15 words per record) and is blocked (30 words per block).

The file must be in EBCDIC characters (90 characters in a record). The program must insert the invalid character by using the four-bit characters 6F in COMP-2.

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  B-6700.
OBJECT-COMPUTER.  B-6700.
INPUT-OUTPUT SECTION.
FILE-CONTROL.  SELECT DKS ASSIGN TO 14 DISK ACCESS SEQUENTIAL.
DATA DIVISION.
FILE SECTION.
FD DKS BLOCK CONTAINS 2 RECORDS.
01 DSKR SZ 90.
01 REDCMP2 COMP-2.
        03 INVLDG.
            05 INVLD PIC 99.
        03 RESTOFIT.
            05 FILLER SZ 178.
WORKING-STORAGE SECTION.
01 EBCRAY SZ 80.
PROCEDURE DIVISION.
PREPARE-DISK-FILE.
    OPEN OUTPUT DKS.
    MOVE " RUN PROG " TO DSKR.
    MOVE "6F" TO INVLDG.
    WRITE DSKR INVALID KEY GO TO XIT.
    MOVE " END. " TO DSKR.
    MOVE "6F" TO INVLDG.
    WRITE DSKR INVALID KEY GO TO XIT.
    CLOSE DKS.
CALL-SYS-WITH-DISK-FILE.
        CALL SYSTEM WITH DKS.
*
*
CALL-SYS-WITH-ARRAY.
    MOVE " RUN PROG;END." TO EBCRAY.
    CALL SYSTEM WITH EBCRAY.
XIT.
    STOP RUN.
```

Figure 7-7. Example of Option 4 CALL Statement

In Options 1 and 2, the actual-parameter-list must consist of a series of data-items, control-items, and expressions optionally separated by commas. In addition to passing arithmetic values, certain kinds of variables may be passed (received) by reference. As a general rule, the kind of actual parameter must not conflict with the corresponding formal parameter.

The formal parameters that can be declared in COBOL along with the corresponding declaration in ALGOL, and the permissable kinds of actual parameters that can be passed are shown in table 7-2.

Table 7-2. Formal and Actual Parameters in COBOL

	COBOL Formal Parameter	ALGOL Formal Parameter	Permissable Actual Parameter
1.	77-level single precision COMP, COMP-1, or COMP-4 item (RECEIVED BY CONTENT)	REAL or INTEGER	Arithmetic-Expression
2.	77-level extended precision COMP, COMP-1, or COMP-5 item (RECEIVED BY CONTENT)	DOUBLE	Arithmetic-Expression
3.	77-level single precision COMP, COMP-1, or COMP-4 item (RECEIVED BY REFERENCE)	REAL or INTEGER	77-level single precision COMP, COMP-1, or COMP-4 item.
4.	77-level extended precision COMP, COMP-1, or COMP-5 item (RECEIVED BY REFERENCE)	DOUBLE	77-level extended precision COMP, COMP-1, or COMP-5 item.
5.	77-level EVENT or LOCK item	EVENT	77-level EVENT or LOCK item
6.	77-level CONTROL-POINT item	TASK	77-level CONTROL-POINT item
7.	77-level INDEX FILE item	DIRECT SWITCH FILE	77-level INDEX FILE item
8.	FILE	FILE	FILE
9.	DIRECT FILE	DIRECT FILE	DIRECT FILE
10.	01-level EVENT or LOCK item	EVENT ARRAY	01-level EVENT or LOCK item
11.	01-level CONTROL-POINT item	TASK ARRAY	01-level CONTROL-POINT item

Table 7-2. Formal and Actual Parameter in COBOL (Cont)

	COBOL Formal Parameter	ALGOL Formal Parameter	Permissable Actual Parameter
12.	01-level COMPUTATIONAL item	ARRAY, INTEGER or REAL	*01-level COMP, COMP-2, DISPLAY, or DISPLAY-1 item
13.	01-level COMP-2 item	HEX ARRAY	*01-level COMP, COMP-2, DISPLAY, or DISPLAY-1 item
14.	01-level DISPLAY-1 item	BCL ARRAY	*01-level COMP, COMP-2, DISPLAY, or DISPLAY-1 item
15.	01-level DISPLAY item	EBCDIC ARRAY	*01-level COMP, COMP-2, DISPLAY, or DISPLAY-1 item

NOTES

1. ASCII ARRAYS are considered the same as EBCDIC ARRAYS for parameter passing purposes.
2. DIRECT ARRAYS may be passed to NON-DIRECT ARRAYS, but not vice versa.
*3. The COBOL compiler will change the character type and length of an array descriptor, if necessary, to match the character type of the formal parameter, for one-dimensional arrays only. The character type of a two-dimensional array must match that of the formal parameter.
4. Any item that may be PASSED BY REFERENCE may be declared GLOBAL.
5. "STACK" ARRAYS (COMP-1, 01-level items) may not be passed as parameters.
6. Conditional expressions may be passed to installation intrinsics having Formal Value Boolean Parameters; any data item may be passed to installation intrinsics having Formal Value Pointer Parameters.
7. A subscripted 01-level item which has an OCCURS clause may be passed to a one-dimensional array.
8. The COBOL compiler reserves the right to prevent elementary 01-level numeric items of any USAGE from being considered "Arrays".

CAUSE

The CAUSE statement is normally used for communication between processes in an asynchronous processing environment.

The format of the CAUSE statement is as follows:

CAUSE [AND RESET] event-identifier-1 [, event-identifier-2]...

The CAUSE statement causes the event specified by event-identifier to be turned on. If the event is then tested in either the same process or any related asynchronous processes, it will cause the TRUE branch to be taken. (Refer to the IF event-identifier statement.) If any of the processes are in a suspended condition because they encountered a WAIT event-identifier statement, they will continue processing. Once the event is caused, it remains on until it is turned off explicitly by the RESET statement.

When AND RESET is specified, all of the event-identifiers are set (i.e. CAUSE) and then RESET thus allowing a process which has been suspended by a "WAIT event-identifier" to continue processing.

Event-identifier must be a properly qualified and subscripted data-name with USAGE EVENT specified

CHECKPOINT

Checkpoint

The CHECKPOINT statement can be used to reduce execution time when it would otherwise be necessary to rerun an entire program.

The CHECKPOINT procedure takes a complete snapshot of the task and stores it on disk. The task can be restarted by utilizing the stored information. If a HALT/LOAD or other system interruption occurs, the Work Flow Management system will automatically restart the job either at the last "no task active" point (see WFM documentation) or at the most recently executed CHECKPOINT statement.

The COBOL syntax to take a checkpoint is as follows:

```
CHECKPOINT
     [ TO   { DISKPACK } ]
            { DISK     }
     [ WITH { PURGE  } ]
            { LOCK   }
     [ ; ON EXCEPTION { statement      } [ ELSE { statement      } ] ]
                      { NEXT SENTENCE }       { NEXT SENTENCE }
```

The DISK-DISKPACK option determines the medium to be used for the checkpoint files.

The PURGE option is used to save only the last checkpoint file.

The LOCK option indefinitely saves all files and can also be used to restart the program even if it has terminated normally. **PURGE** is the default.

If the ON EXCEPTION clause is used, any outcome of a checkpoint except a successful one will cause the statement to be executed (restart is an exception).

After each attempt to take a checkpoint, the CHECKPOINT-STATUS (a special-register) will contain the result of the attempt. CHECKPOINT-STATUS is a global variable, that is, its address resides at Level 2.

The fields may be extracted by use of the MOD and DIV operators in a COMPUTE statement. In the case of a restart, the whole value of CHECKPOINT-STATUS will be negative.

CLOSE

The CLOSE statement is used to indicate that processing of a file has been completed. The file must have been opened previously before a CLOSE statement can be executed.

The format for the CLOSE statement consists of three options which are as follows:

Option 1:

```
CLOSE file-name-1 [ [WITH { NO REWIND }] ]
                  [       { LOCK      }  ]
                  [       { RELEASE   }  ]
                  [       { PURGE     }  ]
                  [       { CRUNCH    }  ]

  [ , file-name-2 [ [WITH { NO REWIND }] ] ] ...
  [               [       { LOCK      }  ] ]
  [               [       { RELEASE   }  ] ]
  [               [       { PURGE     }  ] ]
  [               [       { CRUNCH    }  ] ]
```

Option 2:

```
CLOSE file-name-1 { REEL } [WITH NO REWIND]
                  { UNIT } [FOR REMOVAL   ]

  [ , file-name-2 { REEL } [WITH NO REWIND] ] ...
  [               { UNIT } [FOR REMOVAL   ] ]
```

Option 3:

```
CLOSE HERE file-name [WITH NO REWIND]
```

The REEL and WITH NO REWIND options apply only to files stored on tape.

RELEASE, CRUNCH, and PURGE are Burroughs extensions. In options 1 and 2, the word WITH is optional. CRUNCH is provided for output disk files only.

For input files, any USE declaratives for the ending label are performed when an attempt is made to read beyond the final data record of the file. For output files, the ending label USE routines are performed when the CLOSE statement is executed for the file.

To show the effects of the CLOSE options, each type of file will be discussed separately. The effects of option 1 are as follows:

a. CARD INPUT. The only options allowed are WITH RELEASE and WITH LOCK; however, these are ignored, so the action is the same as a simple CLOSE: the input areas are released and the card reader is returned to the MCP.

b. CARD OUTPUT. Only the options WITH RELEASE and WITH LOCK are allowed; however, these are ignored and the action taken is the same as a simple CLOSE: the output areas are released, the trailer label (if any) is punched, and the unit is returned to the MCP.

c. TAPE INPUT.

1. CLOSE rewinds the tape. It does not release input areas, and the unit remains assigned to the program. Subsequent opening of the file within the same program will always result in accessing the same unit.
2. CLOSE NO REWIND, same as CLOSE except the tape is not rewound.
3. CLOSE LOCK releases the input areas, rewinds the tape, and the MCP marks the unit not ready.
4. CLOSE WITH RELEASE releases the input areas, rewinds the tape, and returns the unit to the MCP.
5. CLOSE PURGE releases the input areas, rewinds the tape, and if a write ring is in the reel, overwrites the label with a scratch label, making the tape a scratch tape.

d. TAPE OUTPUT.

1. CLOSE writes the trailer label (if any), rewinds the tape, and the buffer areas are released. The unit remains assigned to the program.
2. CLOSE 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 buffer areas are released and the unit remains assigned to the program.
3. CLOSE 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 PURGE releases the output areas, writes the trailer label (if any), rewinds the tape, returns the unit to the MCP, and the MCP overwrites the label with a scratch label, making the tape a scratch tape.

The CLOSE options WITH RELEASE, NO REWIND, and PURGE may not be used in conjunction with the REEL option.

e. PRINTER OUTPUT. Only the options WITH LOCK and WITH RELEASE are allowed. However, these have no effect, and the action is always a simple CLOSE: a page is ejected, a trailer label (if any) is written, and the printer is returned to the MCP.

f. DISK FILES. The action 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 disk or pack 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. (See the OPEN statement in this section for further information on new and old files.)

1. CLOSE file-name. For an old file, the file is left in the directory and is available to other programs. Subsequent OPEN OUTPUT will access the same file without resetting EOF pointer.

   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 file-name NO REWIND. Not permitted on DISK files.

3. CLOSE file-name WITH RELEASE.

   For an old file, same as CLOSE file-name.
   For a new file, same as CLOSE file-name PURGE.

4. CLOSE file-name LOCK.

   For an old file, same as CLOSE file-name.
   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.

5. CLOSE file-name PURGE.

   An old file is removed from the DISK and deleted from the directory and may not be reopened.

   A new file will be removed from the DISK. The file may not be opened again by this program for input or I/O.

6. CLOSE file-name CRUNCH.

   CRUNCH is provided only for output disk files. Any unused space at the end of the file is returned to the MCP and the remaining file is then closed as if LOCK had been specified. Later on, the file must not be enlarged.

Option 2 of the CLOSE statement terminates the processing of REELS/UNITS, files and/or removal where applicable.

Use of the FOR REMOVAL phrase is permitted only while the **ANSI74 system** dollar option is set. Refer to Appendix B for a description of the ANSI 74 implementations.

The REEL/UNIT phrase is only used for tape files.

The terms REEL and UNIT are synonymous and completely interchangeable in the CLOSE statement.

TAPE INPUT.

CLOSE file-name REEL allows an end-of-reel condition to be forced. The tape will be rewound and locked; a new reel will be requested by the MCP just as if a normal end-of-reel had occurred. At this point the file is still open and may not be opened in the program. If no more records are to be read from the new reel, it should be closed with release to avoid unnecessary retention of the reel.

TAPE OUTPUT.

CLOSE file-name REEL allows an end-of-reel condition to be forced. Any partial block will be written followed by normal end-of-reel labels (if standard labels are specified). The tape will then be rewound and locked.

Option 3 is an extension to COBOL-68 to permit writing over the last portion of a tape file, or adding to an existing tape file. The operation is as follows:

a. To overwrite the last portion of a file:

   1. OPEN INPUT file-name.
   2. READ file-name. Check RECORD-COUNT, BLOCK-COUNT, or record contents to determine the end of the portion to be retained.
   3. READ file-name. This positions the file at the first record to be overwritten.
   4. When this point is reached, CLOSE HERE file-name WITH NO REWIND. This switches from reading to writing.

5. No OPEN OUTPUT is required, because the only operation allowed after a CLOSE HERE is a WRITE. If an OPEN OUTPUT file-name is given, then the non-fatal attribute error MYUSE (20) will be displayed.

6. WRITE record-name. Successive WRITE statements will overwrite the existing records.

7. CLOSE file-name to write a normal end-of-file on the tape.

b. To add to an existing file:

1. OPEN INPUT file-name.

2. READ file-name.

3. When the AT END clause is executed, CLOSE HERE file-name WITH NO REWIND. This causes the tape to be backspaced over the ending label and the end-of-file mark.

4. WRITE record-name will start adding records to the file immediately following the last data record which previously existed; e.g., if the last record in the file had been record number 5, the first write will write record number 6, and subsequent writes will be to 7, 8, 9, etc.

5. CLOSE file-name to write a normal end-of-file on the tape.

COMPUTE

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

The format for this statement is as follows:

```
COMPUTE identifier-1 [ROUNDED] [, identifier-2 [ROUNDED]] ...

{FROM  }   {identifier-n}
{  =   }   {literal-1   }
{EQUALS}   {formula     }

    [; ON SIZE ERROR statement [ELSE statement]]
```

Literal-1 must be a numeric literal. Each identifier must refer to an elementary numeric item, except identifiers that appear only to the left of:

```
{FROM  }
{  =   }
{EQUALS}
```

may describe data items that contain editing symbols.

The formula option permits the use of any meaningful combination of identifiers, numeric literals, arithmetic operators, and intrinsic functions, parenthesized as required.

The maximum size of each operand is 23 decimal digits.

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

The words FROM and EQUALS are equivalent to each other and to the symbol "=". They may be used interchangeably and the choice is generally made for readability.

If more than one identifier is specified for the result of the operation (that is, preceding FROM, =, or EQUALS), the value of the arithmetic expression is computed, and then this value or the values of literal-1, or identifier-n is stored as the new value of each identifier-1, identifier-2, etc., in turn.

The COMPUTE statement allows the user to combine arithmetic operations without the restrictions on the composition of operands and/or receiving data items imposed by the arithmetic statements ADD, SUBTRACT, MULTIPLY, and DIVIDE. For further information, refer to the discussions of the formulas and the ROUNDED and SIZE ERROR option in this section.

Some examples of the COMPUTE statement are as follows:

```
COMPUTE AMOUNT FROM TONS * COST-PER-TON.
COMPUTE A, B ROUNDED, C EQUALS 487.9563.
COMPUTE X = Y * Z + SQRT (PREV-RES)/W
    ON SIZE ERROR GO TO ERROR-CON ELSE GO TO CNTUE.
```

Either one of the statements shown in the ON SIZE ERROR clause may be conditional or imperative, or the reserved words NEXT SENTENCE may be used instead.

CONTINUE

The CONTINUE statement passes control to a synchronous process that has been previously called and exited via the EXIT PROGRAM RETURN HERE statement, thus allowing the called process to continue without repassing parameters.

The format of the CONTINUE statement is as follows:

CONTINUE control-point-identifier

Control-point-identifier must be the same as in a previously executed CALL statement.

COPY

The COPY statement allows PROCEDURE DIVISION statements contained on a library file to be incorporated into the source program.

The format of the COPY statement is as follows:

```
COPY library-name [FROM seq.no.] [ { THRU     } seq.no. ]
                                 { THROUGH }
    [REPLACING word-1 BY text-1  [, word-2 BY text-2] ... ] .
```

Library-name is the name of a subfile on library. It may be any number of statements, paragraphs, or sections. The subfile may not contain a COPY statement. The COPY statement must be terminated by a period and be the only statement in a sentence.

Library-name may take one of two forms:

a. It may appear as a unique identifier, in which case it becomes the internal name of the library file. This name may be label-equated to another name.
b. It may appear as a non-numeric literal, in which case it is the actual external title of the desired file, (e.g., "A/B/C" for a file titled "A/B/C").

The COPY statement itself is carried over to the symbolic, but the text of the library file does not become part of the symbolic file.

The replacement function is handled internally by the compiler, but no physical change is actually made.

For further information refer to section 8, THE COBOL LIBRARY.

DEALLOCATE

The DEALLOCATE statement may be used to deallocate the storage of record areas. The format of the DEALLOCATE statement is as follows:

DEALLOCATE record-name

The record-name must be an 01 level item that is not redefined and is not described with the SEGMENT clause.

A record-name which is described without a RECORD AREA clause need never be specified in a DEALLOCATE statement, since normal system overlay will release the area of memory used.

The record-name specified must not have a usage of EVENT, LOCK, CONTROL-POINT or COMP-1.

DETACH

The DETACH statement disassociates a procedure from a control-point item or an event item.

The format of the DETACH statement is as follows:

DETACH identifier-1 [, identifier-2] ...

The identifiers used in this statement must be defined as elementary control-point items or the section-name of an interrupt procedure.

If a control-point item is being detached, it must have been previously attached because of the execution of an option 1 CALL statement or a PROCESS statement. The successful execution of this statement terminates a task which was attached to that control-point and had been running. After execution of the DETACH, the control-point item should be tested for a STATUS of -1 prior to the next use of that control-point item. The execution of the program which executed the DETACH continues asynchronously while the detachment is performed.

Similarly, the interrupt procedure being detached must have been attached to an event. If, at the time the detachment occurs, executions of the interrupt procedure have been queued, they will never occur.

DISALLOW

The DISALLOW statement prevents execution of an interrupt procedure which has been attached to an event.

The format of the DISALLOW statement is as follows:

```
DISALLOW  { section-name-1 [, section-name-2] ... }
          { INTERRUPT                             }
```

The section-names specify which interrupt procedure or procedures should not be executed when the attached event(s) is (are) caused. The sections named must be defined as interrupt procedures with the USE statements in their headers. The section-names must be allowed at the time the DISALLOW is executed.

After execution of this statement any potential executions of affected interrupt procedures are queued rather than executed.

DISPLAY

The DISPLAY statement provides for the printing of low-volume data, error messages, and operator instructions on the console.

The format for the DISPLAY statement is as follows:

```
DISPLAY { literal-1    } [, { literal-2    }] ...  [UPON { mnemonic-name }]
        { identifier-1 }    { identifier-2 }            { hardware-name }
```

Each literal may be any figurative constant, except ALL. If a figurative constant is specified as one of the operands, only a single occurrence of the figurative constant is displayed.

When the DISPLAY consists of multiple operands, the data comprising the first operand is DISPLAYed, followed by the data comprising the second operand, and so on. This operation continues until all information is DISPLAYed.

Altogether, no more than 256 characters may be displayed. Identifier-1, identifier-2, ... may be any elementary or group item other than an index data item, index-name, EVENT, LOCK or CP. If an attribute-expression or a special-register is used as identifier-1, identifier-2, ..., the implicit class of the item must be numeric.

Because of the comparatively slow speed of the console, the DISPLAY statement should be used sparingly. Since all operator instruction is thru the console, only information required by the operator should be displayed.

The UPON option is for documentation only. If specified, the hardware-name must be MESSAGE-PRINTER, SPO, KEYBOARD, or DISPLAY-UNIT. Mnemonic-name must be associated with one of the above hardware-names in SPECIAL-NAMES.

DIVIDE

The DIVIDE statement divides one numeric data item into others and sets the values of data items equal to the quotient and remainder.

There are five options for the DIVIDE statement which are as follows:

Option 1:

```
DIVIDE [MOD] {literal-1   }   INTO
             {identifier-1}

        identifier-2 [ROUNDED] [,identifier-3 [ROUNDED]] ...
       [;ON SIZE ERROR statement [ELSE statement]]
```

Option 2:

```
DIVIDE [MOD] {literal-1   }   INTO   {literal-2   }
             {identifier-1}          {identifier-2}

        GIVING identifier-3 [ROUNDED][,identifier-4 [ROUNDED]] ...
       [;ON SIZE ERROR statement [ELSE statement]]
```

Option 3:

```
DIVIDE [MOD] {literal-1   }   BY     {literal-2   }
             {identifier-1}          {identifier-2}

        GIVING identifier-3 [ROUNDED][,identifier-4 [ROUNDED]] ...
       [;ON SIZE ERROR statement [ELSE statement]]
```

Option 4:

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

        GIVING identifier-3 [ROUNDED] REMAINDER
        identifer-4 [ROUNDED] [;ON SIZE ERROR statement [ELSE
        statement]]
```

Option 5:

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

        GIVING identifier-3 [ROUNDED] REMAINDER
        identifier-4 [ROUNDED][;ON SIZE ERROR statement [ELSE
        statement]]
```

Each identifier must refer to a numeric elementary item, except that any identifiers that appear only to the right of the word GIVING may refer to data items that contain editing symbols.

Each literal must be a numeric literal.

The maximum size of each operand is 23 decimal digits. The composite of operands, which is the data item resulting from the superimposition of all receiving data items aligned on their decimal points, must not contain more than 23 digits.

When option 1 is used, the value of identifier-1 or literal-1 is divided into the value of identifier-2. The value of the dividend (identifier-2) is replaced by this quotient; the same is true for identifier-1 or literal-1 and identifier-3, etc.

When option 2 is used, the value of identifier-1 or literal-1 is divided into identifier-2 or literal-2, and the result is stored in identifier-3, identifier 4, etc.

When option 3 is used, the value of identifier-1 or literal-1 is divided by the value of identifier-2 or literal-2, and the result is stored in identifier-3, identifier-4, etc.

Options 4 and 5 are used when a remainder from the division operation is desired, namely identifier-4. A remainder in COBOL is defined as a result of subtracting the product of the quotient and the diviwor from the dividend. If the ROUNDED option is specified, the quotient is rounded after the remainder is determined. The compiler will recognize the ROUNDED option for the REMAINDER clause of the DIVIDE statement only while the B2500 system dollar option is set.

For discussion of the ROUNDED and SIZE ERROR options, see the statement options in this section.

The MOD option can only be used with the B2500 system option set. Specifically, the use of the MOD option will cause the remainder to be placed in identifier-2 of Option 1 and identifier-3 of Options 2 and 3. 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 zero filled. This option provides compatibility with B 3700 COBOL.

Either one of the statements shown in the ON SIZE ERROR clause may be conditional or imperative, or the reserved words NEXT SENTENCE may be used instead.

DUMP

The DUMP statement provides a debugging "snapshot" of specified names in the program.

The format of the DUMP statement is as follows:

```
DUMP { file-name } ( { data-name-1     }  [ , { data-name-2     } ] ... )
     { PRINTER }   { procedure-name }        { procedure-name }
                   { ALL           }         { ALL           }

    paragraph-name-n : { literal     }
                       { data-name-n }  .
```

The DUMP statement must precede the first procedure-name in the PROCEDURE DIVISION. If the program also contains a MONITOR statement, the DUMP statement must follow the MONITOR statement. The word DUMP must begin in margin A and the file-name must be a file assigned to the PRINTER. If no such file exists, the word PRINTER may be substituted. In either case, the file must be OPEN before the first DUMP is actually taken. If PRINTER is specified, the special statement OPEN OUTPUT DIAGNOSTIC must be executed prior to the execution of the first DUMP. Only one DUMP statement is permitted in a program.

The parentheses and the colon must appear as shown in the format. Data-name-1, data-name-2, etc., may not be subscripted, but they may be qualified. They may be group-names or record-names, in which case, the snapshot will actually show all the elementary items contained in the group or record. However, tables may not be DUMPed (see the MONITOR statement).

The operation of the DUMP is as follows. A counter is established, and each time control passes to paragraph-name-n, a one is added to the counter. The counter is then compared with the literal or data-name-n. If the counter is equal or greater, it is reset to zero and a DUMP is executed. In any case, program execution then continues. The use of data-name-n allows the DUMP interval to be varied.

The result of executing a DUMP is a listing on the printer of the data-names appearing in the DUMP statement, and the current value of each one. Procedure-names in the DUMP statement are listed along with a value equal to the number of times the control has been passed to that point in the program. When ALL is specified, then every procedure-name in the program will participate in the DUMP statement. The value of each data-name is shown in edited form if it is numeric, with decimal point and sign. COMPUTATIONAL fields are converted to DISPLAY before being printed.

ENTER

The ENTER statement will transfer control to a procedure which is bound to the COBOL program with the SYSTEM/BINDER. The BINDER is described in a separate document entitled B 7000/B 6000 SYSTEM/BINDER REFERENCE MANUAL, Form No. 5001456.

The format of the ENTER statement is as follows:

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

The ENTER statement is synonymous with option 2 of the CALL statement.

The identifiers in the USING clause of the ENTER statement must be level 77 or 01 data-names with usages of COMP, COMP-1, COMP-2, COMP-4, COMP-5, DISPLAY, DISPLAY-1, ASCII, EVENT, or LOCK.

The procedure that will be entered from the COBOL program must have an external procedure description in the DECLARATIVES, with an accompanying LOCAL-STORAGE entry if parameters are passed.

EXAMINE

The EXAMINE statement is used to replace a specified character and/or to count the number of occurrences of a particular character in a data item.

The format for the EXAMINE statement consists of two options:

Option 1:

```
EXAMINE identifier-1 TALLYING { ALL       } { literal-1    }
                              { LEADING   } { identifier-2 }
                              { UNTIL FIRST }

                [ REPLACING BY { literal-2    } ]
                               { identifier-3 }
```

Option 2:

```
EXAMINE identifier-1 REPLACING { ALL           } { literal-1    }
                               { LEADING       } { identifier-2 }
                               { [UNTIL] FIRST }

                BY { literal-2    }
                   { identifier-3 }
```

The description appearing in the DATA DIVISION for the identifiers used in the EXAMINE statement must be such that USAGE is DISPLAY or DISPLAY-1 (explicitly or implicitly). Literal-1, literal-2, identifier-2, and identifier-3 must consist of a single character belonging to a class consistent with that of identifier-1. A signed numeric literal is not permitted in the EXAMINE statement. Figurative constants automatically represent a single character. Figurative constant ALL may not be used.

Examination of data proceeds as follows:

a. For non-numeric data items, examination starts at the leftmost character and proceeds to the right. Each character in the data item specified by the identifier is examined in turn.

b. If a data item referred to by the EXAMINE statement is numeric, it must consist of numeric characters and may possess an operational sign. Examination starts at the leftmost 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 statement.

The TALLYING option creates an integral count which replaces the value of a special register called TALLY. The count represents the number of the following:

a. Occurrences of literal-1 or identifier-2 when the ALL option is used.
b. Occurrences of literal-1 or identifier-2 prior to encountering a character other than literal-1 or identifier-2 when the LEADING option is used.
c. Characters not equal to literal-1 or identifier-2 encountered before the first occurrence of literal-1 or identifier-2 when the UNTIL FIRST option is used.

When either of the REPLACING options is used, the replacement rules are as follows:

a. When the ALL option is used, then literal-2, identifier-3 is substituted for each occurrence of literal-1, identifier-2.

b. When the LEADING option is used, the substitution of literal-2, identifier-3 terminates as soon as a character other than literal-1, identifier-2 is encountered or the right-hand boundary of the data item is encountered.

c. When the UNTIL FIRST option is used, the substitution of literal-2, identifier-3 terminates as soon as literal-1, identifier-2 is encountered or the right-hand boundary of the data item is encountered.

d. When the FIRST option is used, the first occurrence of literal-1, identifier-2 is replaced by literal-2, identifier-3.

EXECUTE

The EXECUTE statement causes the execution of a separate task, using the procedure referenced. The executed procedure will run independently and asynchronously. An EXECUTE statement has the following form:

```
EXECUTE  control-point-identifier WITH section-name
    [ USING arithmetic-expression-1 [ , arithmetic-expression-2 ]... ]
```

The EXECUTE statement is synonomous with the RUN statement. Refer to the RUN statement.

EXIT

The EXIT statement provides a return mechanism for the PERFORM statement, designates the logical end of a called program, and provides a common end point for a series of procedures in a USE section.

The format of the EXIT statement is as follows:

```
EXIT  [ PROGRAM [RETURN HERE] ]
      [ PROCEDURE             ]
      [ HERE                  ]
```

The EXIT statement must be preceded by a paragraph-name and appear as the only statement in a single sentence within a paragraph. EXIT is normally used in conjunction with procedures referenced by a PERFORM or CALL statement. It allows alternate branch paths within the procedures to rejoin at a common return point, as required by the PERFORM, CALL and USE procedures.

If control reaches an EXIT statement and no associated PERFORM or CALL statement is active, control passes thru the EXIT point to the first sentence in the next paragraph. This cannot happen for USE procedures, as USE sections are considered to be "performed" by the file-handling routines of the MCP, and EXIT statements encountered during execution of a USE section return control to the MCP.

If control reaches an EXIT PROGRAM while under control of a CALL, control is returned to the statement following the CALL in the calling program. If a subsequent CALL statement is executed for the same program, control passes to the first logically executable statement in the called program. When the EXIT PROGRAM RETURN HERE option is used and a CONTINUE statement is executed on the same program, control passes to the statement immediately following the EXIT statement.

The EXIT PROCEDURE statement should be used only for procedures compiled at level 3 or higher. If the procedure has been processed or called as a coroutine, EOT occurs for that stack. If it has been called as a procedure, normal procedure exit occurs back to the statement following the procedure invocation in the calling program.

EXIT

An implicit EXIT PROCEDURE statement is compiled for all procedures compiled at level 3 or higher. The EXIT PROCEDURE statement need not be used when it would be the final statement in the procedure.

The EXIT HERE option may be used as an independent statement. (It does not have the paragraph restrictions.) If the program is under control of a PERFORM statement when EXIT HERE is encountered, the most recently executed PERFORM is exited regardless of whether or not the end of the PERFORM range was reached. If there is not an active PERFORM when the EXIT HERE is executed, control will fall through to the next statement.

GO

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

The format for the GO statement has the following two options:

Option 1:

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

Option 2:

```
GO TO procedure-name-1 [, procedure-name-2] ...
   ,procedure-name-n  DEPENDING ON {formula   }
                                   {identifier}
```

Each procedure-name is the name of a paragraph or a section, or a paragraph-name qualified by a section name.

If procedure-name-1 in option 1 is nct specified, then an ALTER statement referring to this GO TO statement must be executed prior to the execution of this GO TO statement; otherwise, the MCP will terminate the job with an error message referencing the GO paragraph.

If, in option 1, the GO TO statement appears in an imperative sentence, it must appear as the only or last statement in the sequence of imperative statements.

Whenever a GO TO statement (option 1) is executed, control is transferred to procedure-name-1 or to the procedure-name indicated by the last executed ALTER statement.

If the GO TO statement is to be altered when using option 1, then:

a. The GO statement must itself have a paragraph-name, and
b. The GO statement must be the only statement in the paragraph.

In option 2, identifier should be described as a numeric elementary item without any positions specified to the right of the assumed decimal point.

The GO statement in option 2 causes control to be transferred to one of the procedures (procedure-name-1, procedure-name-2,...), depending on the value of the identifier or formula. If the value is either negative, zero, or beyond the range of the procedure-names indicated, control passes to the next statement in the normal sequence of execution. If the value is other than integer, it is truncated to integer.

Example:

```
GO TO AIM, CALC, ERR DEPENDING ON PROC.
```

Figure 7-8 shows the action taken for various values of PROC.

VALUE OF PROC	GO TO
3	ERR
2	CALC
1	AIM
2.4	CALC
-1	NEXT STATEMENT
0	NEXT STATEMENT

Figure 7-8. Result of GO TO ... DEPENDING

Any number of procedure-names may appear in a GO TO procedure-name-list DEPENDING ON ... statement.

Execution of an uninitialized GO TO statement prior to its having been altered will cause an INVALID OPERATOR termination of the program.

IF

The IF statement causes a condition to be evaluated. The subsequent action of the object program depends on whether the value of the condition is true or false.

The format for the IF statement is as follows:

IF condition; [THEN] {statement-1 / NEXT SENTENCE} [; ELSE {statement-2 / NEXT SENTENCE}]

Statement-1 and statement-2 represent either a conditional statement or an imperative statement, and either may be followed by a conditional statement. The semicolons are optional.

The optional use of the reserved word THEN as a delimiter between the conditional expression and the first statement following the IF statement is allowed only while the B2500 system dollar option is set.

The phrase ELSE NEXT SENTENCE may be omitted if it immediately precedes the terminal period of the sentence.

When an IF statement is executed, the following action is taken:

a. If the condition is true, the statements immediately following the condition (represented by statement-1) are executed, and control then passes implicitly to the next sentence unless statement-1 causes some other transfer of control.
b. If the condition is false, either the statements following ELSE are executed or, if the ELSE clause is omitted, the next sentence is executed.

When an IF statement is executed and the NEXT SENTENCE phrase is present, control passes explicitly to the next sentence, depending on the truth value of the condition and the placement of the NEXT SENTENCE phrase in the statement.

IF statements within IF statements may be considered as paired IF and ELSE combinations, proceeding from left to right; thus, any ELSE encountered is considered to apply to the immediately preceding IF that has not already been paired with an ELSE.

ELSE phrases are paired only with IF statements while the ANSI74 or USASI system dollar options are set. Refer to Appendix B for a description of the ANSI 74 implementations.

When control is transferred to the next sentence, either implicitly or explicitly, control passes to the next sentence as written or to a return mechanism of a PERFORM or a USE statement.

IF

The method of evaluating conditional expressions allows early exit once the truth value of the expression has been determined. If the expression contains procedure calls on user intrinsics or makes use of implied subjects, the expression is evaluated fully.

LOCK

The LOCK statement is used in an asynchronous processing environment. It provides the ability for one process to deny related processes access to a particular common storage area until it has unlocked that area. It also permits a process to test a common storage area for a locked condition.

The format of the LOCK statement is as follows:

```
LOCK  { data-name       }  [;  AT LOCKED { statement-1      }  [; ELSE { statement-2      }]]
      { lock-identifier }                { NEXT SENTENCE }            { NEXT SENTENCE }
      { event-identifier}
```

Data-name must be a COMPUTATIONAL or COMP-1 item; and if COMPUTATIONAL, it must be synchronized to a word boundary.

If the AT LOCKED option is specified and the data-name or lock-name is already locked when the LOCK statement is executed, control will pass to the statement following AT LOCKED. If the ELSE option is specified, control passes to the statement following ELSE after the LOCK operation is complete. If the AT LOCKED is not specified, the LOCK statement will continue to try the operation until the LOCK has been successfully completed; that is, until the data-name has been unlocked from another process.

The locking of data-name utilizes the hardware READ-LOCK operator. The locking or unlocking of lock-name or event-name invokes the PROCURE and LIBERATE functions of the MCP.

The AT LOCKED option should be used when locking data-name since there is no priority system with the READ-LOCK operator.

For additional information, refer to the discussion of the UNLOCK statement in this section.

MERGE

The MERGE statement is used to merge up to eight input files into one output file or to make a merged file available to an output procedure. The input files are merged on a set of specified keys.

The format for the MERGE statement is as follows:

```
MERGE   file-name-1      ON
        { ASCENDING  }
        { DESCENDING }   KEY   data-name-1 [, data-name-2] ...
        [ ,ON { ASCENDING  }
              { DESCENDING }   KEY  data-name-3  [, data-name-4] ...] ...
      USING  file-name-2  [, file-name-3 ] ... , file-name-9
    { GIVING  file-name-10                                                  }
    { OUTPUT PROCEDURE IS section-name-1[{ THRU    } section-name-2]        }
                                        { THROUGH }
```

The MERGE verb is considered to be a variation of the SORT verb. Refer to the discussion of the SORT verb (merge mode) in this section for additional merging considerations.

MONITOR

The MONITOR statement provides a debugging "trace" of specified data-names and procedure-names.

The format of the MONITOR statement is:

$$\underline{\text{MONITOR}}\ \left\{\begin{matrix}\text{file-name}\\ \underline{\text{PRINTER}}\end{matrix}\right\}\ \underline{(}\ \left\{\begin{matrix}\text{data-name-1}\\ \text{procedure-name-1}\\ \underline{\text{ALL}}\end{matrix}\right\}\left[,\left\{\begin{matrix}\text{data-name-2}\\ \text{procedure-name-2}\\ \underline{\text{ALL}}\end{matrix}\right\}\right]\ldots\underline{)}$$

The parentheses are required. The MONITOR statement must precede the DECLARATIVES or the first procedure-name in the PROCEDURE DIVISION. If the program also contains a DUMP statement, the MONITOR statement must precede the DUMP. The word MONITOR must begin in margin A. Only one MONITOR statement is permitted in a program.

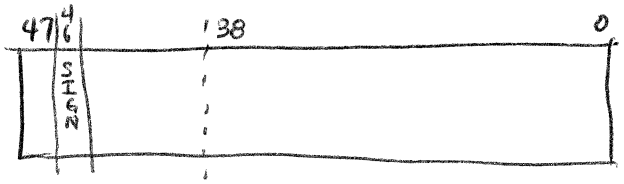
The file-name should be a file assigned to the PRINTER. If no such file exists, the word PRINTER may be substituted.

The MONITOR file may not be explicitly opened after any MONITOR information is written to the file. The file used for writing MONITOR information is created as an ALGOL file to allow the first WRITE to open the file, if it has not already been opened. This is done for the following reasons:

1. The specification in the MONITOR list of the first paragraph-name in the program is made possible.
2. When the MONITOR file is PRINTER, the special statement OPEN OUTPUT DIAGNOSTIC is not necessary.

The data-name(s) may be any names appearing in the DATA DIVISION except file-names and names which require a subscript clause. A table may be monitored by using a group-name or record-name which contains the table.

Whenever an elementary data-name which appears in the MONITOR statement, or is subordinate to a name in the MONITOR statement, is encountered as the result field in a MOVE or arithmetic statement during execution, the name and its new value are listed on the file specified. Procedure-names being monitored are listed when control passes to them, with a count of the number of times control has passed to the procedure-name. In place of, and/or in addition to procedure-name-1, procedure-name-2, the word ALL may be used to indicate that all paragraphs in the program except the first are to be subject to MONITOR action.



MOVE

MOVE

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

The FORMAT for the MOVE statement consists of the following four options:

Option 1:

```
MOVE { special-register     } TO  identifier-2  [, identifier-3] ...
     { attribute-identifier }
     { identifier-1         }
     { literal-1            }
```

Option 2:

```
MOVE { CORRESPONDING } identifier-1 TO identifier-2 [, identifier-3]...
     { CORR          }
```

Option 3:

```
MOVE  [identifier-1 TO identifier-2]
```

Option 4:

```
MOVE  identifier-1 TO identifier-2

      [ { literal-2 } + { literal-3 } + { literal-4 } ]
        { formula-1 }   { formula-2 }   { formula-3 }
```

Identifier-1 and literal-1 represent the sending field; identifier-2, identifier-3 represent the receiving fields. Literal-1 may be any literal or figurative constant consistent with the class of the receiving field.

Options 1 and 2 provide for multiple receiving fields. The data designated by the literal or identifier-1 will be moved first to identifier-2, then to identifier-3, etc. Subscripting or indexing associated with identifier-1 is evaluated only once, immediately before data is moved to the first receiving field. The notes referencing identifier-2 also apply to the other areas.

The result of the statement:

```
MOVE A(SUB) TO SUB, B(SUB)
```

would produce the same result as:

```
MOVE A(SUB) TO temp.
MOVE temp TO SUB.
MOVE temp TO B(SUB).
```

ELEMENTARY MOVES

Any move in which the sending and receiving items are both elementary items is an elementary move. All other moves are defined as group moves. Every elementary item belongs to one of these five categories:

a. Numeric.
b. Numeric Edited.
c. Alphabetic.
d. Alphanumeric.
e. Alphanumeric Edited.

See the PICTURE clause description in Section 6 for a detailed discussion of these categories. Group items, non-numeric literals, and all figurative constants, except ZEROS and SPACES, are classed as alphanumeric. Numeric literals and the figurative constant ZEROS are classed as numeric. The figurative constant SPACES is classed as alphabetic.

Figure 7-9 shows the legality of the various types of elementary moves. The numbers refer to one of the rules listed in the text following.

Category of Sending Field \ Category of Receiving Field		Alphabetic	Alphanumeric and Alphanumeric Edited	Numeric and Numeric Edited
ALPHABETIC		Yes/6	Yes/4	No/1
ALPHANUMERIC		Yes/6	Yes/4	Yes/5
ALPHANUMERIC EDITED		Yes/6	Yes/4	No/1
NUMERIC	INTEGER	No/2	Yes/4	Yes/5
	NON-INTEGER	No/2	No/3	Yes/5
NUMERIC EDITED		No/2	Yes/4	No/1

Figure 7-9. Elementary Moves

Illegal Elementary Moves. The rules governing illegal elementary moves are as follows:

1. A numeric edited item, alphanumeric edited item, SPACES, or an alphabetic item cannot be moved to a numeric or numeric edited item.
2. A numeric literal, ZEROS, a numeric data item, or a numeric edited item cannot be moved to an alphabetic data item.
3. A non-integer numeric literal or a non-integer numeric data item cannot be moved to an alphanumeric or alphanumeric edited data item.

Legal Elementary Moves. The explanation of legal elementary moves is as follows:

4. When an alphanumeric or alphanumeric edited item is a receiving field, justification and any necessary space filling takes place as defined under the JUSTIFIED clause. If the size of the sending field is greater than the size of the receiving field, the excess characters are truncated on the right after the receiving item is filled.

   If the sending field is described as being signed numeric, the operational sign will not be moved. If the sign occupies a separate character position (as in COMP-2), that character will not be moved and the size of the sending field will be considered to be one less than its actual size.

   For example:

   Given these data descriptions:

```
77 S COMP-2 PIC S9999.
77 R PIC X(6).
```

   Then the statements:

```
MOVE -124 TO S.
MOVE S TO R.
```

   will result in R being equal to "0124 "

5. When a numeric or numeric edited item is the receiving field in an elementary move, data is moved algebraically (that is, values are moved, characters are not moved). Therefore, if the data in the sending field is not numeric, zone bits will be stripped and the data will be modified. Alignment by decimal point and any necessary zero-filling takes place as defined under the JUSTIFIED clause, except where zeros are replaced because of editing requirements.

When a signed numeric item is the receiving field, the sign of the sending field is placed in the receiving field. Conversion of the sign representation takes place as necessary. If the sending field is declared unsigned, but contains signed data, the sign moved to the receiving field is unpredictable.

When an unsigned numeric item is the receiving item, the absolute value of the sending item is moved and no operational sign is generated for the receiving item.

When an alphanumeric item is the sending field, data is moved as if the sending item was described as an unsigned numeric integer.

6. When the receiving field is alphabetic, justification and any necessary space filling takes place as defined under the JUSTIFIED clause. If the size of the sending field is greater than the size of the receiving field, the excess characters are truncated on the right, after the receiving field is filled.

GROUP MOVES

A group move is any move in which either the sending field or the receiving field is a group item. Group moves are handled as alphanumeric to alphanumeric moves, regardless of the class of the receiving field and without consideration for the individual elementary or group items contained within either the sending or receiving area.

If the receiving field is a group containing elementary fields described as COMP or COMP-2, the sending field should be identically described as to sign convention, USAGE, and size. Failure to observe this convention may lead to unexpected and/or erroneous results.

Example:

```
01  DISPLAY-AREA.
    05 DA-1 PIC XX.
    05 DA-2 COMP PIC 99.
    05 DA-3 COMP-2 PIC 9999.
01  REDEF REDEFINES DISPLAY-AREA.
    05 R1 PIC X(10).
```

If the figurative constant ZERO or an EBCDIC data item containing zeros is moved to DISPLAY-AREA, REDEF or R1, accesses to the memory locations described by DA-2 and DA-3 will not yield a value of zero. Instead, DA-2 will contain a very small negative floating point number. If the value 1 is added to DA-2, the result will be a value less than 1. The field DA-3 will contain the value 9090 after the above MOVE statement.

The following moves are not allowed:

a. From a group item to an elementary COMP or COMP-1 item.
b. From an elementary COMP or COMP-1 item to a group COMP item.
c. From a numeric data item, special register or attribute to an ASCII item.
d. From a non-numeric literal of any size to any COMP, COMP-1, COMP-4, or COMP-5 data item.

TRANSLATION

Any necessary translation of data from one form of internal representation to another, i.e., BCL to EBCDIC, EBCDIC to hexadecimal, etc., will be done for any elementary or group move in which data is moved non-algebraically. The type of translation depends on the usages of the sending and receiving data items. Data items declared within the sending or receiving fields are not considered.

For example, moving an elementary numeric item of type integer to a group DISPLAY or DISPLAY-1 item causes the absolute value of the elementary item to be converted to characters of the same size as their destination. Then they are placed in their destination, left-justified, with spaces in any character positions to the right.

INDEX DATA ITEMS

An index data item cannot be used as an operand in a MOVE statement. The SET statement must be used to move index data items.

VALID MOVE COMBINATIONS

Figure 7-10 shows the valid combinations of sending and receiving fields permitted in COBOL.

SOURCE ITEM \ DESTINATION ITEM			GROUP					ELEMENTARY																				
								NUMERIC								NUMERIC-EDITED		ALPHA-NUMERIC			ALPHA-NUMERIC EDITED			ALPHA-BETIC		
										INTEGER				NON-INTEGER														
			COMP	COMP-2	DISPLAY-1	ASCII	DISPLAY	COMP-4/COMP-5	DMS FIELD	COMP	COMP-2	DISPLAY-1	DISPLAY	COMP	COMP-2	DISPLAY-1	DISPLAY	DISPLAY-1	DISPLAY	DISPLAY-1	ASCII	DISPLAY	DISPLAY-1	ASCII	DISPLAY	DISPLAY-1	ASCII	DISPLAY
GROUP	COMP		16	16	17	17	17	*	*	*	16	17	17	*	16	17	17	17	17	17	17	17	17	17	17	17	17	17
	COMP-2		16	2	2	2	2	*	*	*	2	2	2	*	2	2	2	2	2	2	2	2	2	2	2	2	2	2
	DISPLAY-1		16	2	2	2	2	*	*	*	2	2	2	*	2	2	2	2	2	2	2	2	2	2	2	2	2	2
	ASCII		16	2	2	2	2	*	*	*	2	2	2	*	2	2	2	2	2	2	2	2	2	2	2	2	2	2
	DISPLAY		16	2	2	2	2	*	*	*	2	2	2	*	2	2	2	2	2	2	2	2	2	2	2	2	2	2
LITERAL	FLOATING-POINT		*	*	*	*	*	5	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
	"UNDIGIT"		*	6	*	*	*	*	*	*	6	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
	NUMERIC INTEGER		*	25	25	25	25	3	7	3	1	1	1	3	1	1	1	20	20	25	25	25	26	26	26	*	*	*
	NUMERIC NON-INTEGER		*	25	25	25	25	3	7	3	1	1	1	3	1	1	1	20	20	*	*	*	*	*	*	*	*	*
	ALPHANUMERIC		*	23	23	23	23	*	*	*	9	9	9	*	9	9	9	21	21	23	23	23	24	24	24	23	23	23
ELEMENTARY	NUMERIC	COMP-4/COMP-5	*	14	14	*	14	19	7	8	12	12	12	8	12	12	12	22	22	14	*	14	15	*	15	*	*	*
		DMS FIELD	*	*	*	*	*	4	7	4	4	4	4	4	4	4	4	22	22	*	*	*	*	*	*	*	*	*
		INTEGER COMP	*	14	14	*	14	19	7	8	12	12	12	8	12	12	12	22	22	14	*	14	15	*	15	*	*	*
		INTEGER COMP-2	16	2	2	2	2	3	7	3	1	1	1	3	1	1	1	20	20	14	*	14	15	*	15	*	*	*
		INTEGER DISPLAY-1	16	2	2	2	2	3	7	3	1	1	1	3	1	1	1	20	20	14	*	14	15	*	15	*	*	*
		INTEGER DISPLAY	16	2	2	2	2	3	7	3	1	1	1	3	1	1	1	20	20	14	*	14	15	*	15	*	*	*
		NON-INTEGER COMP	*	*	*	*	*	18	7	8	12	12	12	8	12	12	12	22	22	*	*	*	*	*	*	*	*	*
		NON-INTEGER COMP-2	16	2	2	2	2	3	7	3	1	1	1	3	1	1	1	20	20	*	*	*	*	*	*	*	*	*
		NON-INTEGER DISPLAY-1	16	2	2	2	2	3	7	3	1	1	1	3	1	1	1	20	20	*	*	*	*	*	*	*	*	*
		NON-INTEGER DISPLAY	16	2	2	2	2	3	7	3	1	1	1	3	1	1	1	20	20	*	*	*	*	*	*	*	*	*
	NUMERIC-EDITED	DISPLAY-1	16	2	2	2	2	*	*	*	*	*	*	*	*	*	*	*	*	2	2	2	13	13	13	*	*	*
		DISPLAY	16	2	2	2	2	*	*	*	*	*	*	*	*	*	*	*	*	2	2	2	13	13	13	*	*	*
	ALPHA-NUMERIC	DISPLAY-1	16	2	2	2	2	10	11	10	9	9	9	10	9	9	9	21	21	2	2	2	13	13	13	2	2	2
		ASCII	16	2	2	2	2	10	11	10	9	9	9	10	9	9	9	21	21	2	2	2	13	13	13	2	2	2
		DISPLAY	16	2	2	2	2	10	11	10	9	9	9	10	9	9	9	21	21	2	2	2	13	13	13	2	2	2
	ALPHA-NUMERIC EDITED	DISPLAY-1	16	2	2	2	2	*	*	*	*	*	*	*	*	*	*	*	*	2	2	2	13	13	13	2	2	2
		ASCII	16	2	2	2	2	*	*	*	*	*	*	*	*	*	*	*	*	2	2	2	13	13	13	2	2	2
		DISPLAY	16	2	2	2	2	*	*	*	*	*	*	*	*	*	*	*	*	2	2	2	13	13	13	2	2	2
	ALPHA-BETIC	DISPLAY-1	16	2	2	2	2	*	*	*	*	*	*	*	*	*	*	*	*	2	2	2	13	13	13	2	2	2
		ASCII	16	2	2	2	2	*	*	*	*	*	*	*	*	*	*	*	*	2	2	2	13	13	13	2	2	2
		DISPLAY	16	2	2	2	2	*	*	*	*	*	*	*	*	*	*	*	*	2	2	2	13	13	13	2	2	2

Figure 7-10. Valid MOVE Statement Combinations

The following rules describe the valid combinations of sending and receiving fields shown in figure 7-10. These rules apply in addition to the standard alignment rules for destinations as explained in the discusssion of the JUSTIFIED clause.

* **Illegal Move**

1. Numeric-decimal move; absolute value moved if destination is described as unsigned. Completion of operation guarantees that zones and sign of destination, if any, are valid. If digits are greater than 9, they may not be moved unchanged from the source.
2. Alphanumeric move; left justified with truncation or space fill (zero fill if the destination is COMP-2) on the right (except if the destination is described with the JUSTIFIED clause). If the usage of the source and destination are not the same, translation occurs using the standard MCP translate tables. The source is considered as having a category of alphanumeric. If the source is a group item, the destination is considered as having a category of alphanumeric (any editing or decimal point is ignored).
3. Numeric move; the decimal source is converted to binary. "UNDIGITS" existing in the source are changed to values less than 10. If the destination is COMP-4 or COMP-5, the source value will be approximated in binary floating-point.
4. Numeric move; the bit pattern of the source is considered to be an unsigned integer operand, and is extended to double precision binary or converted to decimal if necessary.
5. Numeric move; decimal value converted to nearest binary floating-point approximation, and adjusted to single or double precision.
6. "UNDIGIT" move; right justified, zero fill on left.
7. Numeric move; the source is converted to a binary integer if necessary. The destination is considered to be an unsigned binary integer operand. High-order bits of the source may be truncated.
8. Numeric move; the binary source value is adjusted to the precision and scale of the destination. If the source value is floating-point, the source value is integerized.
9. Numeric move; the source is considered as a numeric unsigned integer, and is moved as described in rule-1.
10. Numeric move; the source is considered as a numeric unsigned integer, and is moved as described in rule-3.

11. Numeric move; the source is considered as a numeric unsigned integer, and is moved as described in rule-7.
12. Numeric move; the binary source is converted to decimal.
13. Alphanumeric-edited move; the source is considered as having a category of alphanumeric, and is moved as in rule-2, except that editing also takes place.
14. Numeric to alphanumeric move; the source is assumed to contain valid numeric data and is converted, if necessary, into an intermediate unsigned numeric integer data item whose length is equal to the number of decimal places described in the picture clause of the source and whose usage is the same as the destination usage. This intermediate data item is then considered as having a category of alphanumeric and is moved as described in rule-2.
15. Numeric to alphanumeric-edited move; the move is done according to rule-14, except that editing also takes place.
16. Group computational move; the source is considered as having a category of alphanumeric. The bit pattern of the source is transferred, left justified (or justified right, if the destination is so described), to the destination, with zero fill.
17. Group computational move; the move is done according to rule-16, except that space fill occurs.
18. Numeric move; a single precision source value is extended to double precision if the destination is COMP-5. A double precision source value is set to single precision if the destination is COMP-4. The source value is then divided by a power of ten corresponding to its scale factor and stored into the destination in floating-point.
19. Numeric move; a single precision source value is extended to double precision if the destination is COMP-5. A double precision source value is set to single precision if the destination is COMP-4.
20. Numeric-edited decimal move; the move is done according to rule-1, except that editing also occurs.
21. Numeric-edited decimal move; the source is considered as a numeric unsigned integer, and is moved as described in rule-20.
22. Numeric-edited move; the binary source is converted to decimal and edited.
23. Alphanumeric move; standard justification and space (or zero) fill. The source characters are considered as being in the character set of the destination.

24. Alphanumeric move; the move is done according to rule-23, except that editing also takes place.
25. Alphanumeric move; the source is considered as being a non-numeric literal, and any sign or decimal point is ignored. The move is then done according to rule-23.
26. Alphanumeric move; the move is done according to rule-25, except that editing also takes place.

Option 2:

When Option 2 is used, selected items within identifier-1 are moved, with any required editing, to selected areas within identifier-2. Identifier-1 and identifier-2 must be group items. Items are selected by matching the data-names of items defined within identifier-1 with like data-names of areas defined within identifier-2, according to the rules specified in the discussion of the CORRESPONDING option. The resulting operation on each of the sets of matched data items proceeds as if an Option 1 MOVE had been specified.

Options 3 and 4:

Options 3 and 4 of the MOVE verb are extensions to COBOL-68. They allow bit manipulation and character manipulation. In options 3 and 4 the left and right bracket symbols are required. In option 4 the colons are required.

In option 3 the sending field item (identifier-1) must be a DISPLAY, DISPLAY-1, or COMP-2 item of any size or class. This is a "blind move" of the first six characters of identifier-1 into the low-order six characters of identifier-2. If identifier-1 is less than six characters long, then only the size of identifier-1 is moved; but the data is right-justified with zero fill in the high-order bits of identifier-2.

In option 4, both identifier-1 and identifier-2 must be COMP or COMP-1 in USAGE. Both must be single precision. This is a move of bits from identifier-1 into identifier-2, with only the indicated bits of identifier-2 being changed. Literal-2 or formula-1 represents the location in identifier-1 from which the transfer begins, i.e., the source-bit. Literal-3 or formula-2 represents the location in identifier-2 at which the transfer begins, i.e., the destination-bit. Literal-4 or formula-3 represents the number of bits to be transferred.

The bits of a B 7000/B 6000 word are numbered 47 thru 0, from left to right. A computational item represents such a word. Therefore, only 0 thru 47 may be used as values for source-bit or destination-bit. Source-bit or destination bit minus the number of bits must not be less than -1.

Examples of options 3 and 4:

```
MOVE [CUST-NAME TO DISK-ADDR]
```

would move the first six characters of CUST-NAME to the COMPUTATIONAL item DISK-ADDR without converting to binary.

```
MOVE A-AND-B-BOTH TO A-ONLY [39:19:20]
MOVE A-AND-B-BOTH TO B-ONLY [19:19:20]
```

These statements would unpack a COMPUTATIONAL word which actually contained two 20-bit fields.

```
MOVE B-ONLY TO A-AND-B-BOTH
MOVE A-ONLY TO A-AND-B-BOTH [19:39:20]
```

would repack the fields.

MULTIPLY

The MULTIPLY statement causes numeric data items to be multiplied, and sets the value of data items equal to the results.

The format for the MULTIPLY statement has two options which are as follows:

Option 1:

```
MULTIPLY {literal-1   } BY
         {identifier-1}
   identifier-2  [ROUNDED] [ , identifier-3  [ROUNDED] ] ...
  [; ON SIZE ERROR  statement  [ELSE  statement] ]
```

Option 2:

```
MULTIPLY {literal-1   } BY {literal-2   } GIVING
         {identifier-1}    {identifier-2}
   identifier-3   [ROUNDED]  [ , identifier-4  [ROUNDED] ] ...
  [;  ON SIZE ERROR  statement  [ELSE  statement] ]
```

Each identifier must refer to a numeric elementary item, except in option 2. In this option, any identifiers that appear only to the right of the word GIVING may refer to items that contain editing symbols.

Each literal must be a numeric literal.

The maximum size of each operand is 23 decimal digits. The composite of operands, which is that data item resulting from the superimposition of all the receiving data items aligned on their respective decimal point must not contain more than 23 digits.

When option 1 is used, the value of identifier-1 or literal-1 is multiplied by the value of identifier-2. The value of the multiplier (identifier-2) is replaced by this product; similarly this is true for identifier-1 or literal-1 and identifier-3, etc.

When option 2 is used, the value of identifier-1 or literal-1 is multiplied by identifier-2 or literal-2, and the result is stored in identifier-3, identifier-4, etc.

For a discussion of the ROUNDED and ON SIZE ERROR options, see statement options discussion.

Either one of the statements shown in the ON SIZE ERROR clause may be conditional or imperative, or the reserved words NEXT SENTENCE may be used instead.

OPEN

OPEN

The OPEN statement initializes the processing of both input and output files. It performs the reading and checking of labels on input, the writing of labels on output, the execution of applicable USE routines, and other input/output functions.

The formats for the OPEN statements are as follows:

Option 1:

```
OPEN { INPUT  file-name-1 [ { WITH LOCK  [ACCESS] } ]
                           { WITH NO REWIND       }
       [ file-name-2 [ { WITH LOCK  [ACCESS] } ] ]                ...
                     [ { WITH NO REWIND      } ]
     { OUTPUT file-name-3  [WITH NO REWIND]
       [,file-name-4  [WITH NO REWIND]]  ...
     { I-O          }
     { INPUT-OUTPUT } [file-name-5 [, file-name-6] ...
     { O-I          }
     { EXTEND       [file-name-7 [, file-name-8] ...                }
```

Option 2:

```
OPEN { INPUT  } file-name-7 REEL-NUMBER { literal   }
     { OUTPUT }             { data-name }
```

At least one of the options, INPUT, OUTPUT, I-O, or O-I must be specified; however, there may be no more than one instance of each option. These options may appear in any order.

The I-O option pertains to disk files existing prior to this opening and to data communication (REMOTE) files.

An OPEN statement must be applied to all files except sort-files and must be executed prior to the first READ, WRITE, or SEEK for a file. Failure to do so will result in an error condition at object program time. Also, the file must be opened before it can be closed, and a second OPEN for a file cannot occur without an intervening CLOSE.

The MCP provides complete file-handling capabilities. It maintains an I/O assignment table which shows which file is mounted on each peripheral device, and which program is currently using each device. When a tape is mounted or a card deck readied in the reader, the MCP reads the label on the front of the file and enters the information in the I/O assignment table. Tapes which contain a write ring and on which the retention period has expired (see SAVE-FACTOR, Section 6) are marked as scratch and will be assigned to output files as required. All other tapes are marked as input and may be accessed only with OPEN INPUT. Thus, a tape (or disk) file may be OPENed OUTPUT, written on, closed and then opened INPUT, provided a SAVE-FACTOR greater than zero is specified.

When an OPEN INPUT statement is executed, the MCP searches in the I/O assignment table or DISK directory for the file specified. If the file is found, it is opened, and any associated USE declaratives are executed. If the file is not in the table or directory, a message to the operator is displayed and the program is suspended until one of the following conditions is met:

a. The file appears in the table or directory.
b. The file is an optional file and the operator indicates that the file is not present (i.e., the OF message).
c. The operator indicates another file as being the one desired (i.e., the IL message).
d. The program is discontinued.

When an OPEN OUTPUT statement is executed for a file, the MCP searches the I/O assignment table or disk directory for available disk space or a peripheral of the type desired, writes the label as specified by the program, and executes any declaratives for the file. If the MCP is unable to find a peripheral or enough disk as required for the file, a message to the operator is typed and the program is suspended until the operator makes available whatever is required or until he discontinues the program.

The I-O options permits opening of a disk or data communication (REMOTE) file for both input and output. If the file is assigned to DISK the file must be present on disk at the time the file is opened.

In option 1, when using the OPEN statement with the LOCK or LOCK ACCESS options, the B2500 system dollar option must be set. When LOCK or LOCK ACCESS is specified, the EXCLUSIVE file attribute is set.

When using the O-I option in option-1, the O-I option will give the same update action that input-output does; however, a new file will be created. The use of this option requires that the B2500 system option be set.

The EXTEND option in option 1 is an ANSI74 system dollar option which exists for sequentially organized files. The ANSI74 system dollar option must be set when using this option.

The EXTEND option is used to position an already-existing file immediately following the last logical record of that file. Subsequent WRITE statements referencing the file will add records to the file as though the file has been opened with the output phrase. Refer to Appendix B for a description of the ANSI 74 implementations.

When a DISK file is opened for OUTPUT and no file size was specified in the SELECT clause for that file, the MCP will use the file size of an existing file of the same name or a default value as discussed in the SELECT clause in section 6.

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.

Opening of subsequent reels of labeled multireel tape files is handled automatically by the system and requires no special consideration by the programmer.

When checking or writing the first label, the user's beginning label subroutine is executed if one is specified by a USE statement.

If an INPUT file is designated as OPTIONAL in the FILE-CONTROL paragraph of the ENVIRONMENT DIVISION and the file is not present (i.e., the OF message is typed by the operator), the first READ for that file will cause the AT END for that file to be executed.

The REVERSED option can only be used with files assigned to TAPE and cannot be used with multiple-reel files. The subsequent READ statements for that file will make the data records of the file available in reverse order, i.e., starting with the last data record.

When neither the REVERSED nor the NO REWIND option is specified, execution of the OPEN statement causes the file to be positioned at its beginning.

Option 2 allows a program to process a file starting at other than the first reel. This can be useful in master file searching and in program restarts.

PERFORM

The PERFORM statement is used to execute one or more procedures, either a specified number of times or until a condition is satisfied, and then to return control to the normal sequence.

The format for a PERFORM statement includes the following four options:

Option 1:

```
PERFORM  procedure-name-1  [ { THROUGH } procedure-name-2 ]
                           { THRU    }
```

Option 2:

```
PERFORM  procedure-name-1  [ { THROUGH } procedure-name-2 ]
                           { THRU    }

         { identifier-7 }
         { integer-1    } TIMES
         { formula-1    }
```

Option 3:

```
PERFORM  procedure-name-1  [ { THROUGH } procedure-name-2 ]
                           { THRU    }

    UNTIL condition-1
```

Option 4:

```
PERFORM  procedure-name-1  [ { THROUGH } procedure-name-2 ]
                           { THRU    }

    VARYING { identifier-1 } FROM { identifier-2 }
            { index-name-1 }      { literal-2    }
                                  { formula-2    }
                                  { index-name-2 }

        BY { identifier-3 } UNTIL condition-1
           { literal-3    }
           { formula-3    }

  [ AFTER { identifier-4 } FROM { identifier-5 }
          { index-name-4 }      { literal-5    }
                                { formula-5    }
                                { index-name-5 }

        BY { identifier-6 } UNTIL condition-2 ] ...
           { literal-6    }
           { formula-6    }
```

Each procedure-name is the name of a section, paragraph, or a paragraph qualified by a section name.

Each identifier represents a numeric elementary item. Identifier-7 must be described as an integer.

Each literal represents a numeric literal.

When the PERFORM statement is executed, control is transferred to the first statement of procedure-name-1. An automatic return to the statement following the PERFORM statement is established as follows:

a. If procedure-name-1 is a paragraph-name and procedure-name-2 is not specified, then the return occurs after the last statement of procedure-name-1.

b. If procedure-name-1 is a section name and procedure-name-2 is not specified, then the return occurs after the last statement of the last paragraph in procedure-name-1.

c. If the procedure-name-2 is specified and it is a paragraph name, then the return occurs after the last statement of the paragraph.

d. If the procedure-name-2 is specified and it is a section name, then the return occurs after the last sentence of the last paragraph in the section.

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 the procedure named procedure-name-1 and ending with the execution of the procedure named procedure-name-2. In particular, GO TO and PERFORM statements may occur between procedure-name-1 and the end of procedure-name-2. No particular sequential relationship is required to exist between procedure-name-1 and procedure-name-2. There may be more than one logical path of program control through the performed range of procedures. A common method, though not a required one, of documenting the terminal paragraph of a performed range of procedures is through the use of the EXIT statement.

If control passes these procedures by means other than a PERFORM statement, control passes through the last statement of the procedure to the following statement, unless a PERFORM statement is executed during execution of these procedures.

An implicit return mechanism is established at the end of a performed range of procedures and is activated by the execution of a PERFORM statement. Program control reaching an active return mechanism will always return to the activating PERFORM statement. A return mechanism permanently deactivates itself by transferring program control back to a PERFORM statement. An active return mechanism is temporarily deactivated by the execution of a PERFORM statement. Program control will always pass through a non-active return mechanism to the next executable statement following the PERFORM range.

Because the return control information is placed in the stack, rather than 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 permanent deactivization of an active return mechanism causes the last return mechanism temporarily deactivated to again become active, allowing "overlapping" PERFORM ranges, or two or more PERFORM ranges that have a common exit point, to logically execute the same as disjoint PERFORM ranges.

Transferring program control, by means of a "GO" statement, from a range of procedures being executed under control of a PERFORM statement does not cause the return mechanism to be deactivated. Subsequently, transferring program control back into the PERFORM range will cause control to return to the PERFORM statement, provided that the return mechanism is still active. Repeatedly branching from a PERFORM range without allowing control to ever reach an active return mechanism may cause a STACK OVERFLOW due to the fact that each execution of a PERFORM puts this mechanism in the stack and expects it to be deleted by execution of the final statement in procedure-name-2. In an input or output procedure, care must be taken to ensure that no more than 31 PERFORM's are active (not yet exited) when a RETURN or RELEASE is executed. If 32 or more PERFORMS are still active when a RETURN or RELEASE are executed, the program will be terminated with INVALID INDEX.

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. When the TIMES option is used, the procedures are performed the number of times specified for that execution by the initial value of identifier-7 or integer-1. When the PERFORM statement is executed, the value of integer-1 must be positive. If the initial value of identifier-7 is negative or zero, control passes immediately to the statement following the PERFORM statement. Following the execution of the

procedures the specified number of items, control is transferred to the statement following the PERFORM statement. During execution of the PERFORM statement, reference to identifier-7 will not alter the number of times the procedures are to be executed.

Option 3 is the UNTIL option. The specified procedures are performed until the condition specified by the UNTIL option is true. At this time, control is transferred to the statement following the PERFORM statement. If the condition is true at the time that the PERFORM statement is encountered, the specified procedure is not executed.

Option 4 is the VARYING option. This option is used to augment the value of one or more identifiers or index-names in an orderly fashion during the execution of a PERFORM statement. Every reference to identifier as the object of the VARYING and FROM (starting value) phrases also refers to index-names. When index-name appears in the FROM phrase, identifier, when it appears in an associated VARYING or AFTER phrase, is initialized according to the rules of the SET statement; subsequent augmentation is as described below.

In option 4, when one identifier is varied, identifier-1 is set equal to the current value of identifier-2, literal-2, or formula-2. If the condition is false, the sequence of procedures, procedure-name-1 thru procedure-name-2, is executed once. The value of identifier-1 is augmented by the specified increment or decrement (identifier-3), and condition-1 is evaluated again. The cycle continues until this expression is true; at this point, control passes to the statement following the PERFORM statement. If the condition is true at the beginning of execution of the PERFORM, control passes directly to the statement following the PERFORM statement. Figure 7-11 illustrates the logic of the PERFORM statement when one identifier is varied.

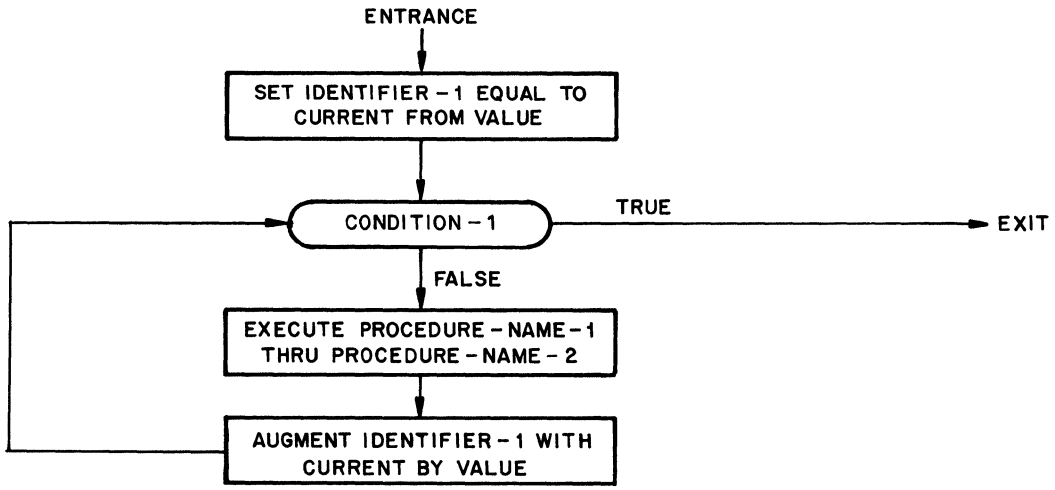


Figure 7-11. PERFORM Statement Varying One Identifier

In option 4, when two identifiers are varied, identifier-1 and identifier-4 are set to the current value of identifier-2 and identifier-5, respectively. At the start of the PERFORM statement, condition-1 is evaluated; if true, control is passed to the statement following the PERFORM statement; if false, condition-2 is evaluted. If condition-2 is false, procedure-name-1 thru identifier-6, and condition-2 is evaluated again. The cycle of execution and augmentation continues until this condition is true. When condition-2 is true, identifier-4 is set to the current value of identifier-5; identifier-1 is augmented by identifier-3, and condition-1 is re-evaluated. The PERFORM statement is completed if condition-1 is true; if not, the cycles continue until condition-1 is true.

Figure 7-12 illustrates the logic of the PERFORM statement when two identifiers are varied.

During the execution of the procedures associated with the PERFORM statement, any change to the VARYING variable (identifier-1 and index-name-1), the BY variable (identifier-3), the AFTER variable (identifier-4 and index-name-4), or the FROM variable (identifier-2, index-name-2, identifier-5 and index-name-5) will be taken into consideration and will affect the operation of the PERFORM statement.

When two identifiers are varied, identifier-4 goes thru a complete cycle (FROM, BY, UNTIL) each time identifier-1 is varied.

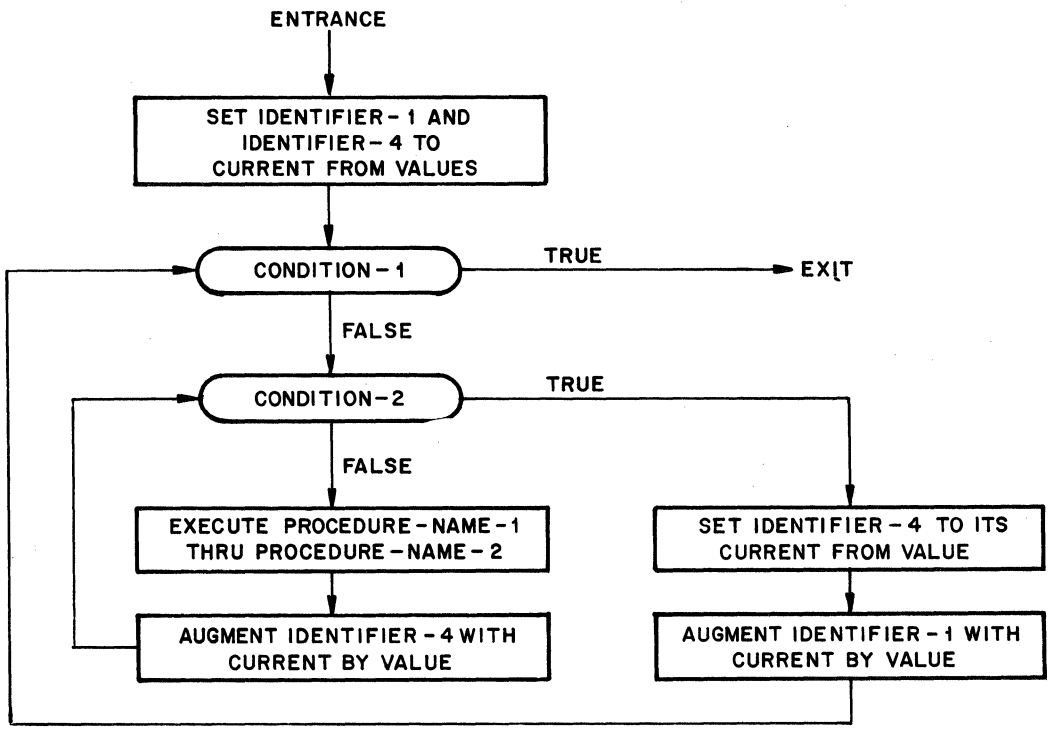


Figure 7-12. PERFORM Statement Varying Two Identifiers

At the termination of the PERFORM statement, identifier-4 contains the current value of identifier-5. Identifier-1 has a value that exceeds the last used setting by an increment or decrement, as the case may be, unless condition-1 was true when the PERFORM statement was entered, in which case identifier-1 contains the current value of identifier-2.

For three identifiers, the mechanism is the same as for two identifiers except that identifier-7 goes through a complete cycle each time identifier-4 is augmented by identifier-6 or literal-6, which in turn goes through a complete cycle each time identifier-1 is varied.

Figure 7-13 illustrates the logic of the PERFORM statement when three identifiers are varied.

After the completion of option 4, identifier-4 and identifier-7 contain the current value of identifier-5 and identifier-8, respectively. Identifier-1 has a value that exceeds its last used setting by one increment or decrement value, unless condition-1 is true when the PERFORM statement is entered, in which case identifier-1 contains the current value of identifier-2.

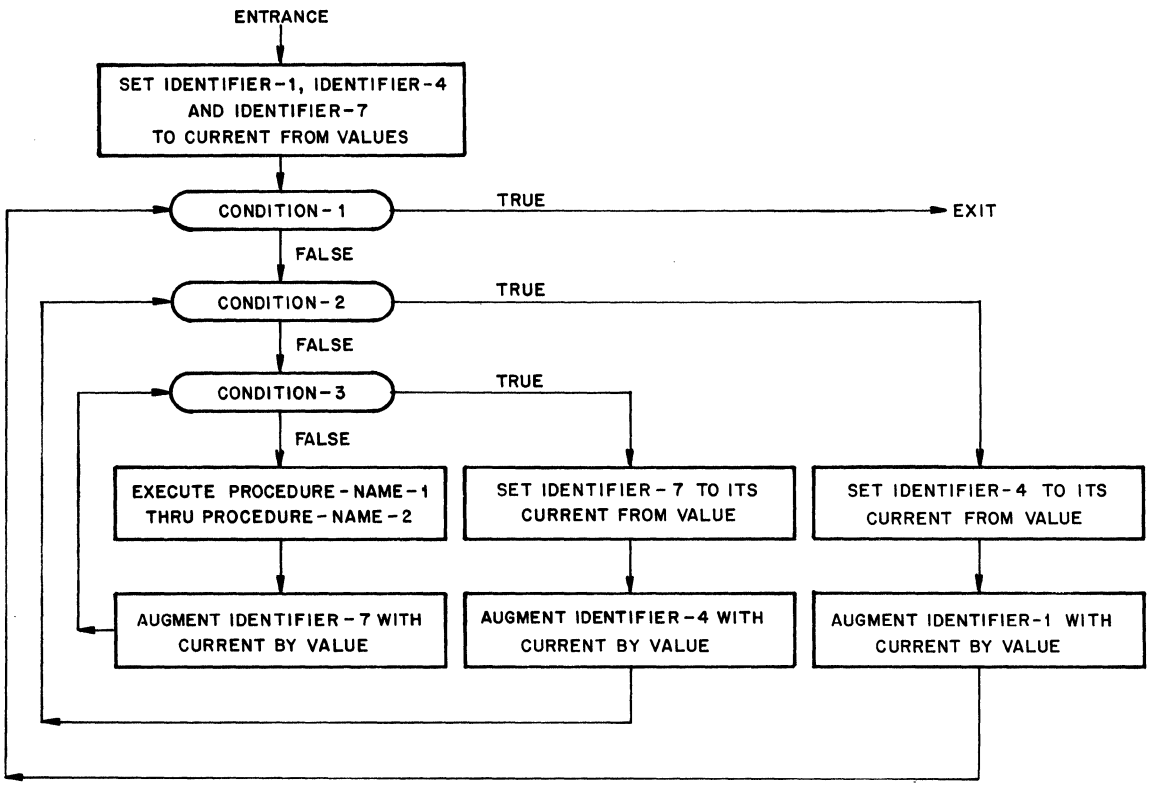


Figure 7-13. PERFORM Statement Varying Three Identifiers

PROCESS

The PROCESS statement initiates the parallel execution of another task.

The format of the PROCESS statement is as follows:

```
PROCESS  control-point-identifier  WITH section-name
   [USING  actual-parameter-list]
```

The syntax of the PROCESS statement is the same as that for the CALL statement. The difference between the two statements is that CALL initiates serial processing of another program while PROCESS initiates parallel (or asychronous) processing of another program.

NOTE: The actual-parameter-list must consist of a series of data-items, control-items, and expressions, optionally separated by commas. In addition to passing arithmetic values, certain kinds of variables may be passed (received) by reference. As a general rule, the kind of actual parameter must not conflict with the corresponding formal parameter.

The PROCESS statement creates a dependent process as a separate task. Unlike the separate task initiated by a RUN statement, the task initiated by a PROCESS statement is dependant on the initiator. If the initiator terminates prior to termination of the dependent process, a critical block exit will occur.

A dependent process must be bound to a host program if the dependent process contains data areas described with the GLOBAL or OWN clauses. A bound dependent process is generally more efficient than an unbound dependent process.

A dependent process runs in its own stack.

Refer to the discussion of the CALL statement for an explanation of syntax and semantics.

READ

The READ statement makes the following items available to the program:

a. For sequential file accessing, the READ statement makes available the next logical record from an input file or an input-output file, and also allows performance of a specified statement when END-OF-FILE is detected.

b. For random accessing, the READ statement makes available a specific record from a mass storage file, and also allows performance of a specific statement if the value of the associated actual key data-name is found to be invalid (out of range).

The READ statement has the following three options:

Option 1:

```
READ file-name RECORD [INTO identifier]
   ;AT END statement [ELSE statement ]
```

Option 2:

```
READ file-name RECORD [INTO identifier]
   ;INVALID KEY statement [ELSE statement ]
```

Option 3:

```
READ file-name [KEY IS formula] INTO identifier
    [USING event-identifier] [ON EXCEPTION statement  [ELSE statement]]
```

An OPEN statement for the file must be executed prior to the execution of the first READ for that file.

In the INTO identifier option, identifier must be a name in WORKING STORAGE or an output record. The operation will be identical to a READ statement without the INTO option followed by a MOVE. When the INTO identifier option is used, the file-name RECORD is still available in the input record area. The file-name in this case must not represent a sort-file. The INTO phrase must not be used when the input file contains logical records of various sizes as indicated by their record descriptions.

Option 1 is used for all serial files, e.g., tape, card, and disk files accessed sequentially. Option 2 is used for disk files accessed randomly and for all data communication (REMOTE) files.

If a file described with the OPTIONAL clause is not present at the time the file is opened, then at the time of the execution of the first READ statement for that file, the AT END condition will occur.

If, during execution of a READ statement in option 1, the logical end of the file is reached and an attempt is made to READ that file, the statement specified in the AT END phrase is executed. The logical end of the file is specified as the last record written (if serial) or as the record with the highest record number ever written (if random). Any further attempt to READ the file once an END-OF-FILE condition has occurred will cause the AT END statement to be executed unless a SEEK statement is used to point to a valid record location or a CLOSE and an OPEN are executed.

End-of-reel conditions (excluding END-OF-FILE) on tape input files are handled automatically and need not be of concern to the programmer, except on DIRECT I/O. On a DIRECT file, these conditions must be handled by the program.

The following action, performed by the B 7000/B 6000 file control routine, takes place at end-of-reel when the tape mark is sensed (unless the file used DIRECT I/O):

a. If labels are STANDARD, the regular end-of-reel checking for the close of an input reel is performed along with any USE routines specified.
b. The next reel of tape is opened; if labels are STANDARD, beginning-of-reel checks are performed, including any specified USE routines.
c. The first record on the new reel is made available to the program.

If a read error is detected during a READ, the MCP will attempt to successfully reread the record. If the record cannot be read correctly, control will be transferred to the applicable USE AFTER ERROR procedure and the record in question is made available to the program.

The occurrence of a parity error without a USE procedure results in the error message PARITY ERROR NO LABEL, followed by job termination.

Error termination also results if a READ statement references a file that no longer is OPEN.

Option 2 is used for random disk files. The READ statement implicitly performs the functions of the SEEK statement (using the actual key specified by the programmer) unless a SEEK statement for the record specified by the current value of the actual key has been executed prior to the READ statement. If the value of actual key is outside the limits of the file, as specified in the FILE-LIMITS clause, the ASSIGN clause, or by the EOF pointer in the disk directory, the INVALID KEY clause will be executed. Successive reads may be executed without changing the value of the actual key, and data will be moved each time to the record area of the file.

Option 3 of the READ statement provides user control of the I/O operation. This option allows the programmer to initiate the I/O operation and return immediately to the statement following the READ. The I/O operation and the user program then run in parallel until the I/O comes to completion or the user program encounters a WAIT statement associated with that READ. THE ON EXCEPTION phrase may be used with option 3 of the READ; however, if this option is included it will cause loss of asynchronous processing with the I/O, since the I/O operation must come to completion before it can be determined if an exception condition occurred. This can be avoided by using the ON EXCEPTION option of the WAIT statement or by testing one or more direct attributes.

The file-name must be described as a DIRECT file or as a DIRECT SWITCH file. (See discussion of DIRECT SWITCH files under USAGE IS INDEX FILE.)

The KEY IS option may be used to override the ACTUAL KEY specification in the FILE-CONTROL paragraph for randomly accessed DIRECT disk files.

The option 3 READ statement places the record directly into working storage; no record description (buffer area) is associated with file-name. The INTO portion of the statement specifies where the record is to be placed. Identifier may be any 01 level entry in the WORKING-STORAGE section which specifies the RECORD AREA clause.

Event-identifier, if used, is a link between the system and the program. When the system completes the I/O operation it notifies the program by causing the event-identifier to happen (an implicit CAUSE statement). The event-identifier specified for the READ must be turned off or RESET before it is used in a subsequent I/O operation; otherwise an error will occur. Refer to the RESET, WAIT, and IF event-identifier statements in this section.

When using option 3 of the READ, the user must provide his own check for the end-of-file or invalid key condition. This can be accomplished by testing the direct attribute IOEOF logically following receipt of the I/O COMPLETE event.

While the ANSI74 system dollar option is set, ELSE phrases will be paired only with IF statements. In particular, ELSE phrases will not be paired with AT END, INVALID KEY, and ON EXCEPTION clauses while the ANSI74 system dollar option is set.

Figure 7-14 illustrates the use of DIRECT I/O in COBOL.

Burroughs COBOL CODING FORM

PROGRAM: DIRECT I/O IN COBOL | REQUESTED BY | PAGE 1 OF 1
PROGRAMMER | DATE | IDENT. 73 80

```
PAGE NO. 1 3  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     SELECT F1    ASSIGN TO DIRECT DISK.
02     SELECT F2    ASSIGN TO DIRECT DISK.
03
04
05   77  RL    PIC 9999 COMP-1.
06   77  E1    USAGE EVENT.
07   77  V2          PIC 9(112)    COMP-2   VALUE 0.
08   77  K1          PIC 9(6)      COMP-1   VALUE 1.
09   01  A     SIZE 1 TO 96 CHARACTERS DEPENDING ON RL    RECORD AREA.
10
11
12
13         OPEN OUTPUT F2.          OPEN INPUT F1.
14         SET A(IOMASK) TO V2.
15
16   L.    MOVE 96 TO RL.
17         READ F1 KEY IS K1 INTO A USING E1.
18         WAIT E1.                 RESET E1.
19         IF A(IOEOF) = VALUE TRUE CLOSE F2 WITH LOCK STOP RUN.
20         MOVE A(IOCHARACTERS) TO RL.
           WRITE F2 FROM A.
           WAIT A ON EXCEPTION DISPLAY "OUTPUT ERROR" STOP RUN.
           ADD 1 TO K1.             GO TO L.
```

Figure 7-14. Direct I/O in COBOL

RELEASE

The RELEASE statement transfers records to the initial phase of the SORT operation.

The format for the RELEASE statement is as follows:

```
RELEASE record-name [FROM identifier ]
```

RELEASE is used within the range of an INPUT PROCEDURE associated with a SORT statement. Any other use of a RELEASE statement will lead to unpredictable results.

Record-name must be the 01 name of the record in its associated sort-file.

If the FROM clause is used, the operation will be identical to a MOVE followed by a RELEASE without the FROM clause. The information in the record area is available.

After the RELEASE is executed, record-name is no longer available. The execution of a RELEASE statement causes record-name (after identifier has been moved to it in the FROM option) to be transferred to the initial phase of a SORT. When control passes from the INPUT PROCEDURE, the file consists of all records placed in it by the execution of RELEASE statements. No OPEN, CLOSE, READ, WRITE or USE statements may be given for the sort-file.

RESET

The RESET statement is used as part of the user's control of DIRECT input-output operations and also for communication between processes in an asynchronous processing environment.

The format for the RESET statement is as follows:

```
RESET event-identifier-1 [, event-identifier-2]...
```

The RESET statement causes the event specified by event-identifier to be turned off. For inter-process communication, refer to the CAUSE statement. For DIRECT files, refer to the discussion of option 3 of the READ statement or option 5 of the WRITE statement.

Event-identifier must be a properly qualified and subscripted data-name with USAGE IS EVENT specified.

RETURN

The RETURN statement is used to obtain sorted records from the final phase of the SORT operation.

The format for the RETURN statement is as follows:

```
RETURN file-name RECORD [INTO identifier ]
          ;AT END statement [ELSE statement ]
```

File-name must be a sort-file with a sort-file description (SD) in the DATA DIVISION. RETURN can only be used within the range of an OUTPUT PROCEDURE. Any other use of a RETURN statement will lead to unpredictable results.

The execution of the RETURN statement causes the next record in sorted order (according to the keys listed in the SORT statement) to be made available for processing in the record area associated with the sort-file. No OPEN, READ, WRITE, CLOSE, or USE statements may be given for the sort-file.

The storage area associated with the identifier and the storage area which is the record area associated with file-name must not be the same storage area.

If the INTO clause is used, the operation is identical to a RETURN without the INTO option followed by a MOVE. The file-name RECORD is still available in the input area.

After execution of the AT END statement, no RETURN statement may be executed within the current OUTPUT PROCEDURE. The results of such an error will be unpredictable.

Any reference to the record area associated with the specified file-name after the AT END condition has occurred will cause an error termination of the program.

RUN

The RUN statement allows a program to initiate another program. Once a program is initiated by execution of a RUN statement, it is executed independently of the calling program.

The format of the RUN statement is as follows:

```
RUN control-point-identifier WITH section-name
    [USING arithmetic-expression-1
        [,arithmetic-expression-2] ... ]
```

The section-name must be the name of a use procedure declared to be external. All parameters must be passed by value, that is, RECEIVED BY CONTENT. Only arithmetic values may be passed or received, since only formal parameters having arithmetic properties can be described in the syntax. The formal parameters, to which the values of the arithmetic expressions are passed, must be described as single or extended precision 77-level **COMP, COMP-1, COMP-4, or COMP-5** items and should have a RECEIVED BY CONTENT clause, even though RECEIVED BY CONTENT is the default. The compiler makes any adjustments, if necessary, to truncate extended precision values to single precision, or extended single precision values to insure that the value passed has the same precision as the corresponding formal parameters. All values are passed with a scale of zero, regardless of the scale of the corresponding formal parameter, and may be passed as normalized values.

The RUN statement creates an independent process which does not in any way, share the resources of the initiator and thus may continue running after the termination of the initiator.

An independent process must not be bound to a host program nor may any of its data be declared with the GLOBAL or OWN phrases. Independent procedures must be compiled at level 2.

SEARCH

The SEARCH statement is used to search a table for a table element that satisfies the specified condition and to adjust the associated index-name to indicate that table element.

The two formats of the SEARCH statement are:

Option 1:

```
SEARCH identifier-1 [VARYING {identifier-2 }]
                             {index-name-1 }

 [; 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 identifier-1

  [; AT END imperative-statement-1]

   ; WHEN condition-1 {imperative-statement-2 }
                      {NEXT SENTENCE          }
```

The SEARCH statement is to be used if a linear search of a table is desired.

Identifier-1 must not be subscripted or indexed, but its description in the DATA DIVISION must contain an OCCURS clause and an INDEXED BY clause. Identifier-1 in Option 2 must also be described with the KEY IS phrase in its OCCURS clause.

Identifier-2, when specified, must be described as USAGE IS INDEX or as an integer numeric elementary item, i.e., without any positions to the right of the assumed decimal point.

The WHEN condition in Option 2 should be constructed so that each key item is referenced.

If the VARYING index-name-1 phrase is specified and index-name-1 appears in the INDEXED BY clause of identifier-1, that index-name is used for this search. If this is not the case or if the VARYING identifier-2 phrase is specified, the first (or only) index-name given in the INDEXED BY clause of identifier-1 is used for the search. In addition, the following operations will occur:

a. If the VARYING index-name-1 phrase is used and index-name-1 appears in the INDEXED BY clause 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 identifier-1 is incremented.

b. If the VARYING identifier-2 phrase is specified, and identifier-2 is an index data item, then the data item referenced by identifier-2 is incremented by the same amount as, and at the same time as, the index associated with identifier-1 is incremented. If identifier-2 is not an index data item, the data item referenced by identifier-2 is incremented by the value one (1) at the same time as the index referenced by the index-name associated with identifier-1 is incremented.

In Option 1, if the varying phrase is not used the index-name that is used for the search operation is the first (or only) index-name that appears in the INDEXED BY clause of identifier-1. Any other index-names for identifier-1 remain unchanged.

Condition-1, condition-2, etc., may be any condition as described at the beginning of this chapter. The search starts with the current index-name setting as follows:

a. If at the start of execution of the SEARCH statement, the index-name associated with identifier-1 contains a value that corresponds to an occurrence number that is greater than the highest permissible occurrence number for identifier-1, the search is terminated immediately. Then if the AT END clause is specified, imperative-statement-1 is executed; otherwise, control passes to the next sentence.

b. If at the start of execution of the SEARCH statement, the index-name associated with identifier-1 contains a value that corresponds to an occurrence number within the range of permissible occurrence numbers for identifier-1, the SEARCH statement operates by evaluating the conditions in the order of their appearance, making use of the index settings, wherever specified, to determine the occurrence of those items to be tested. If none of the conditions are satisfied, the index-name for identifier-1 is incremented to obtain reference to the next occurrence. The process is then repeated using the new index-name settings, unless the new value of the index-name settings for identifier-1 corresponds to a table-element outside the permissible range of occurrence values, in which case the search terminates,

passing control to the AT END clause or next sentence. 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.

A diagram of the operation of the SEARCH statement containing two WHEN phrases is shown in figure 7-15.

In general, if an imperative-statement of an AT END clause or a WHEN clause does not terminate with a GO TO statement, then, after execution of the imperative-statement, control will pass to the next sentence.

If identifier-1 is a data-item subordinate to a data-item that contains an OCCURS clause (providing for a two- or three-dimensional table), an index-name must be associated with each dimension of the table using the INDEXED BY phrase of the OCCURS clause. Only the setting of the index-name associated with identifier-1 (and the data-item identifier-2 or the index-name-1, if present) is modified by the execution of the SEARCH statement. To search a multi-dimensional table, it is necessary to execute a SEARCH statement several times. Prior to each execution of a SEARCH statement, SET statements must be executed whenever index-names must be adjusted to appropriate settings.

In an Option 2 SEARCH,

a. The data in the table should be ordered in the same manner as described in the KEY clause associated with the description of identifier-1.
b. The contents of the key(s) referenced in the WHEN clause must be sufficient to identify a unique table element.
c. The primary purpose of the WHEN clause is to provide an algorithm for determining when to stop searching the table.

In Option 2, the index-name that is used for the search operation is the first (or only) index-name that appears in the INDEXED BY clause of identifier-1. Any other index-names for identifier-1 remains unchanged. The index-name to be used for the SEARCH is initialized to 1 and then incremented as discussed for an option 1 SEARCH.

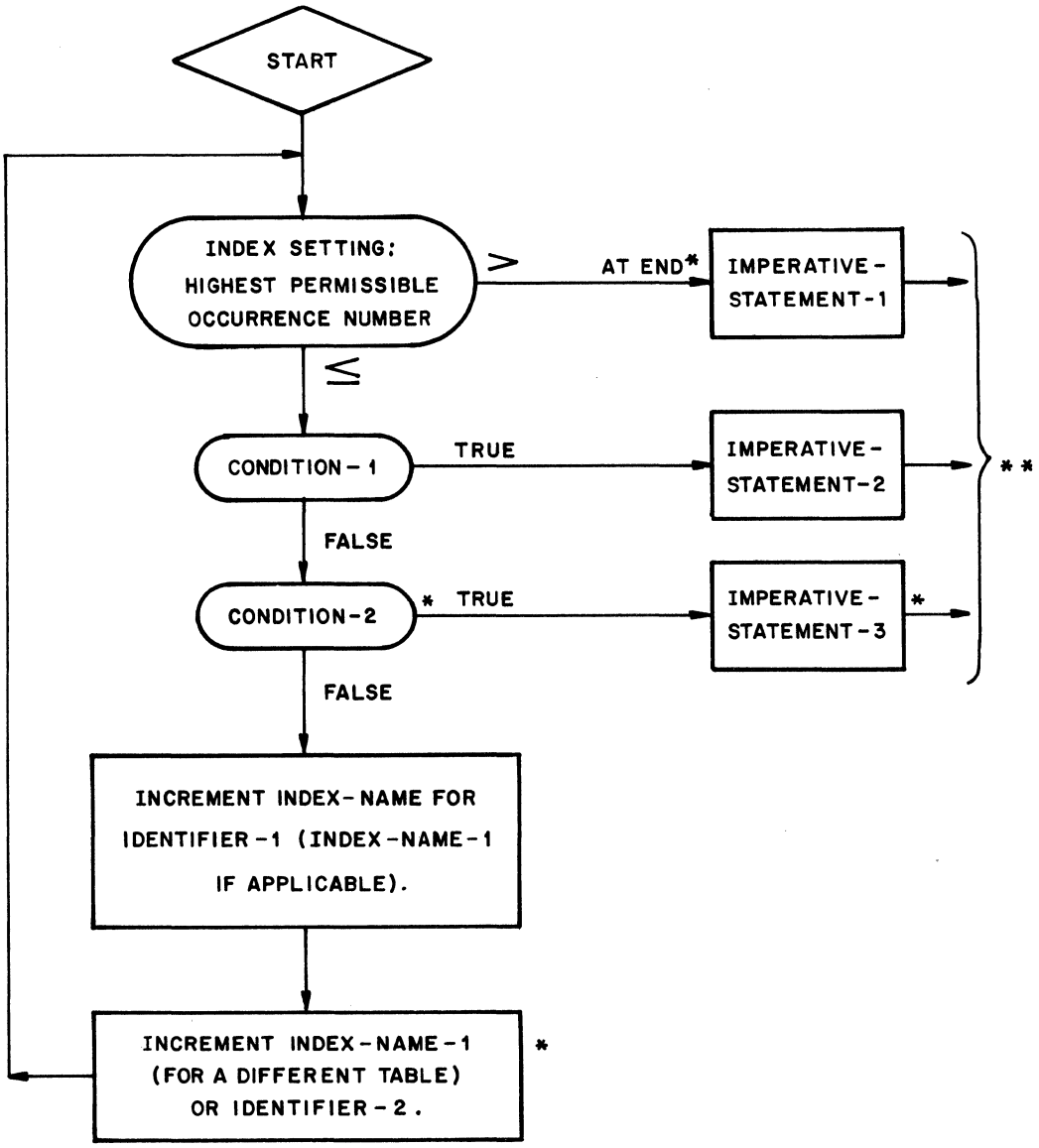


Figure 7-15. SEARCH Operation Containing Two WHEN Phrases

*This option is included only when specified in the SEARCH statement.

**This control transfer is to the next executable sentence unless the imperative-statement ends with a GO TO statement.

SEEK

The SEEK statement initiates the accessing of a disk file record for subsequent reading and/or writing.

The format for the SEEK statement is as follows:

```
SEEK  file-name  RECORD [WITH KEY CONVERSION] ...
```

The SEEK statement, with the KEY CONVERSION clause, will cause the USE FOR KEY CONVERSION routine associated with the file to be invoked prior to executing the SEEK. The KEY CONVERSION clause can only be specified while the B2500 system dollar option is set.

The SEEK statement may only be used on disk files. The programmer must specify the ACTUAL KEY for random files and may specify an ACTUAL KEY for sequential files. The value of the ACTUAL KEY at the time the SEEK statement is executed determines the record sought.

If the value of the ACTUAL KEY is invalid (i.e., outside of the bounds of the file or not within the limits specified by the FILE-LIMITS clause), the statement in the AT END or INVALID KEY clause of the next executed READ or WRITE statement for that file is executed.

Two SEEK statements for the same disk file may logically follow each other.

A SEEK statement used in a serial file serves to reset the file to the record specified by ACTUAL KEY.

A SEEK statement used in a random file serves to pre-locate a record. This allows concurrent I/O operations.

SET

The SET statement establishes reference points or offsets operations by setting index-names associated with table elements. The SET statement is also used to specify a value for a file attribute or task attribute.

The two formats of the SET statement are:

Option 1:

```
SET { identifier-1 }   [, { identifier-2 }]...  TO { identifier-3 }
    { index-name-1 }       { index-name-2 }        { index-name-3 }
                                                   { integer-1    }
```

Option 2:

```
SET  index-name-4  [, index-name-5]  ...  { UP BY   }  { identifier-4 }
                                          { DOWN BY }  { integer-2    }
```

All references to identifier-1 and index-name-1 apply equally to identifier-2 and index-name-2, respectively.

All identifiers must name either index data items, or elementary items described as an integer, except that identifier-4 must not name an index data item. When integer-1 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 phrase of the OCCURS clause.

If index-name-3 is specified, the value of the index before the execution of the SET statement must correspond to an occurrence number of an element in the associated table.

If index-name-1, index-name-2 is specified, the value of the index after the execution of the SET statement must correspond to an occurrence number of an element in the associated table. The value of the index associated with an index-name after the execution of a SEARCH or PERFORM statement may be undefined.

In option 1, the following action occurs:

a. Index-name-1 is set to a value causing it to refer to the table element that corresponds in occurrence number to the table element referenced by index-name-3, identifier-3, or integer-1. If identifier-3 is an index data item, or if index-name-3 is related to the same table as index-name-1, no conversion takes place.

b. If identifier-1 is an index data item, it may be set equal to either the contents of index-name-3 or identifier-3 where identifier-3 is also an index data item; no conversion takes place in either case.

c. If identifier-1 is not an index data item, it may be set only to an occurrence number that corresponds to the value of index-name-3. Neither identifier-3 nor integer-1 can be used in this case.

d. The process is repeated for index-name-2, identifier-2, etc., if specified. Each time, the value of index-name-3 or identifier-3 is used as it was at the beginning of the execution of the statement. Any subscripting or indexing associated with identifier-1, etc., is evaluated immediately before the value of the respective data item is changed.

In option 2, the contents of index-name-4 are incremented (UP BY) or decremented (DOWN BY) by a value that corresponds to the number of occurrences represented by the value of integer-2 or identifier-4; thereafter, the process is repeated for index-name-5, etc. Each time the value of identifier-4 is used as it was at the beginning of the execution of the statement.

Data in the figure 7-16 represents the validity of various operand combinations in the SET statement. The parenthetical comment references the lettered paragraphs above.

	RECEIVING ITEM		
SENDING ITEM	INTEGER DATA ITEM	INDEX-NAME	INDEX DATA ITEM
Integer Literal	No (c)	Valid (a)	No (b)
Integer Data Item	No (c)	Valid (a)	No (b)
Index-Name	Valid (c)	Valid (a)	Valid (b)*
Index Data Item	No (c)	Valid (a)*	Valid (b)*
*No conversion takes place.			

Figure 7-16. SET Statement Operand Combinations

SORT

The SORT statement is used to create a sort-file by executing input procedures or by transferring records from another file, to sort the records in the sort-file on a set of specified keys and, in the final phase of the operation, to make available each record from the sort-file, in sorted order, to an output procedure or to an output file.

The format for the SORT statement is as follows:

```
SORT  file-name-1 [ { PURGE } ON ERROR ]  ON  { ASCENDING }  KEY data-names
                    { RUN   }               { DESCENDING }
                    { END   }

[,data-name-2 ] ...

      [ , ON { ASCENDING } KEY data-name-3 [, data-name-4 ] ... ] ...
             { DESCENDING }

      { USING  file-name-2 [ { LOCK    } ] }
      {                      { PURGE   } }
      {                      { RELEASE } }
      {
      { INPUT PROCEDURE IS section-name-1 [ { THROUGH } section-name-2 ] }
      {                                     { THRU    }                  }

      { GIVING  file-name-3 [ { LOCK    } ] }
      {                     { RELEASE } }
      {
      { OUTPUT PROCEDURE IS  section-name-3 [ { THROUGH } section-name-4 ] }
      {                                     { THRU    }                  }

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

      [ DISK SIZE formula-2 { WORDS   } ]
      [                     { MODULES } ]

      [ RESTART IS { formula-3   } ]
      [            { data-name-5 } ]
      [            { literal-1   } ]
```

More than one SORT statement may appear in a program. SORT statements can appear anywhere in the PROCEDURE DIVISION, except within INPUT and OUTPUT procedures associated with a SORT statement or within the DECLARATIVES.

File-name-1 must have a sort-file description in the DATA DIVISION. File-name-2 and file-name-3 must be described in a file description entry, not in a sort-file description entry.

The keys are listed from left to right in the SORT statement in order of significance, without regard to how they occur within the record. Data-name-1 is the major key followed in descending significance by additional keys if any.

a. When a SORT file-name-1 PURGE clause is used, PURGE implies that on a parity error, the bad record will be skipped and SORTing will continue.
b. When a SORT file-name-1 RUN clause is used, RUN implies that on a parity error, the bad record will be SORTed.
c. When a SORT file-name-1 END clause is used, END implies that the SORT will be DSed on encountering a parity error.
d. When an ASCENDING clause is used, the sorted sequence will be from the lowest value of key to highest value.
e. When a DESCENDING clause is used, the sorted sequence will be from the highest value of key to lowest value.
f. Both ASCENDING and DESCENDING ekys can be used in one SORT statement.
g. When a USING file-name-2 LOCK clause is used, LOCK implies that the file will be LOCKed at the end of the SORT.
h. When a USING file-name-2 PURGE clause is used, PURGE implies close with PURGE.
i. When a USING file-name-2 RELEASE clause is used, RELEASE implies close with RELEASE.
j. When a GIVING file-name-3 LOCK clause is used, LOCK implies that the file will be closed with LOCK at the end of the SORT.
k. When a GIVING file-name-3 RELEASE clause is used, RELEASE implies close with RELEASE.

NOTE: Items a thru c and g thru k can be implemented only when the B2500 system dollar option is set.

Every data-name appearing in the KEY clause must be described under the DATA DIVISION entry for the sort-file-name, and these KEY items are subject to the following rules:

a. They must not be variable length items.
b. No key may be more than 65,535 characters in length. (Should it be necessary to sort on a key which exceeds this limit, the definition and use of subitems will bypass the restriction.)
c. All numeric elementary items are compared algebraically, with all negative values considered lower than positive values. Anything else is compared as alphanumeric. DISPLAY keys are compared according to the EBCDIC collating sequence. DISPLAY-1 keys are translated to EBCDIC and compared according to the EBCDIC collating sequence. COMP-2 keys are compared as 4-bit data characters. ASCII is compared without translation. When two or more records are identical on all keys specified, the order of these equal records in the sorted output will not always be the order of their entry into the sort. When records which are equal on all keys must be returned from the sort in some predictable order, an INPUT PROCEDURE may be used to append a record number to each record and that record number may then be specified as the most minor key for the sort.
d. KEY items cannot themselves contain nor can they be subordinate to entries which contain the OCCURS clause.
e. All key(s) must be in the same location within all records. An INPUT PROCEDURE may be used to modify any records which do not meet these standards.
f. The data names may be qualified. When a data-name specified for a key is not unique, qualification by file-name-1 is implied.

If an INPUT PROCEDURE is specified, control is passed to the INPUT PROCEDURE before file-name-1 is sequenced by the SORT statement. The INPUT PROCEDURE must consist of one or more sections that appear contiguously in a source program and do not form a part of any output procedure. The compiler inserts a return mechanism at the end of the last section in the INPUT PROCEDURE, and when control passes from the last statement in the INPUT PROCEDURE, the records which have been released to file-name-1 will be sorted.

The use of an INPUT PROCEDURE allows the programmer to select, create or modify the input records to the SORT. The INPUT PROCEDURE must include at least one RELEASE statement in order to transfer records to the sort-file; at least one record must be released. Control must not be passed to an INPUT PROCEDURE, except when a related SORT statement is being executed, since the RELEASE statement(s) will have no meaning unless under the control of a sort. The following are certain rules and restrictions which must be followed when writing an INPUT PROCEDURE:

a. The INPUT PROCEDURE must not contain any SORT or MERGE statements.
b. The remainder of the PROCEDURE DIVISION must not contain any transfers of control to points inside the INPUT PROCEDURE; i.e., ALTER, GO, and PERFORM statements in the remainder of the PROCEDURE DIVISION are not permitted to refer to procedure-names within the INPUT PROCEDURE. Also, while the INPUT PROCEDURE may perform a program portion outside of itself, control must always return to the INPUT PROCEDURE.

If the USING file-name-2 option is specified, this implies that all the records in file-name-2 are transferred automatically to file-name-1. At the time of execution of the SORT statement, file-name-2 must not be OPEN. The SORT statement will automatically perform the necessary OPEN, READ, USE, and CLOSE functions for file-name-2. File-name-2 must not be a sort-file description.

File-name-1 and file-name-2 must have the same character size since no input translation will be performed by the sort. Translation can be done in the input procedure.

If an OUTPUT PROCEDURE is specified, control passes to it after file-name-1 has been sequenced by the SORT statement. The output procedure must consist of one or more sections that appear contiguously in a source program and do not form part of any input procedure. The compiler inserts a return mechanism at the end of the last section in the OUTPUT PROCEDURE. When control passes from the last statement in the OUTPUT PROCEDURE, the return mechanism will provide for termination of the sort and then will send control to the next statement after the SORT statement. Before entering the OUTPUT PROCEDURE, the SORT procedure reaches a point at which it can select the next record in sorted order when requested. The RETURN statements in the OUTPUT PROCEDURE are the requests for the next record.

The OUTPUT PROCEDURE must include at least one RETURN statement in order to make sorted records available for processing. Control must not be passed to the OUTPUT PROCEDURE, except when a related SORT statement is being executed, because the RETURN statement does not have meaning unless under control of a

sort. The OUTPUT PROCEDURE can consist of any procedure needed to select, modify, or copy the records which are being returned one at a time in sorted order from the sort-file. The following are certain rules and restrictions which must be followed in writing an OUTPUT PROCEDURE:

a. The OUTPUT PROCEDURE must not contain any SORT or MERGE statements.
b. The remainder of the PROCEDURE DIVISION must not contain any transfers of control to points inside the OUTPUT PROCEDURE; ALTER, GO, and PERFORM statements in the remainder of the PROCEDURE DIVISION are not permitted to refer to procedure-names within the OUTPUT PROCEDURE. While an OUTPUT PROCEDURE may PERFORM a program portion outside of itself, control must always return to the OUTPUT PROCEDURE.
c. No RETURN statement can be executed after control has been transferred to the AT END statement.

If the GIVING option has been used, this means that all the sorted records in file-name-1 are automatically transferred to file-name-3 as the implied output procedure for this SORT statement. At the time of execution of the SORT statement, file-name-3 must not be OPEN. File-name-3 is automatically opened before the records are transferred, and a CLOSE file-name-3 is executed automatically after the last record in the sort-file is returned. File-name-3 must have a file description, not a sort-file description, in the DATA DIVISION.

File-name-1 and file-name-3 must have the same character size, since the sort will not perform translation on output; the user can include appropriate translation in the OUTPUT PROCEDURE.

If the MEMORY SIZE and/or DISK SIZE clauses are present in a SORT statement, they take precedence over the values stated in the OBJECT-COMPUTER paragraph of the ENVIRONMENT DIVISION. The presence of a MEMORY SIZE clause for a SORT statement has no effect on the compilers algorithm for estimating core require ments for the object program.

The RESTART clause is used to communicate to the SORT the type of restart desired. Literal-1 must be a numeric literal. Data-name-5 must reference an elementary integer numeric data item. The RESTART values and the SORT are discussed in the B 7000/B 6000 System Operation Guide, Volume Number One, Form No. 5001563.

SORT

The COBOL sort package is designed to sort a file or merge a number of files into a single file of ordered records. The package operates in four modes: DISK-ONLY MODE, TAPE-ONLY MODE, ITD SORT MODE, and MERGE MODE. The mode is selected by the programmer as follows:

a. DISK-ONLY MODE:

1. COBOL mode selection statement:

   SELECT file-id ASSIGN TO SORT DISK.

2. System configuration. The amount of disk specified by the DISK SIZE clause should be 1.5 to 2 times the file size.

3. Characteristics of disk-only sort:
   (a) The disk-only mode is the fastest of the three sort modes.
   (b) The disk-only mode requires less intervention on the part of the operator to set up the system for sorting.
   (c) The amount of data that can be sorted is restricted by the amount of available disk.
   (d) If amount of available disk is inadequate, the sort will be suspended with the "NO USER DISK" message on the SPO. When this happens either of the following can be done:
      (1) The sort can be terminated and another sort mode selected.
      (2) Permanent files on the disk can be moved to tape. The "OK" message will then cause sorting to resume.

b. TAPE-ONLY MODE:

1. COBOL mode selection statement:
   SELECT file-ID ASSIGN TO integer-1 SORT-TAPES.
2. System configuration. A minimum of three tape drives is required for the sort-tapes. The maximum number of sort-tapes allowed is eight.
3. Characteristics of the tape-only sort:
   (a) Allows for multi-reel sort scratch tapes.
   (b) The input file to be sorted may be indefinite in length (since the scratch tapes may be multi-reel).

c. ITD SORT MODE:

1. COBOL mode selection statement:
SELECT file-ID ASSIGN TO SORT DISK AND integer-1 { TAPES / SORT-TAPES }

2. System configuration: A minimum of three tape drives are required for the sort-tapes. The maximum number of sort-tapes allowed is eight.

3. Characteristics of the ITD sort mode:

(a) If the amount of available disk is sufficient, the ITD sort will operate in the disk mode only.

(b) If the amount of disk is insufficient, sorting will be automatically switched to the tape mode.

(c) While operating in the ITD sort mode, the disk is used in conjunction with core to edvelop strings on the scratch tapes up to 500 times longer than is possible with the tape-only mode. This can reduce sort times by a large factor, compared to what can be obtained by operating in the tape-only mode.

(d) The scratch tapes, if used, may be multi-reel, thus allowing the sorting of an indefinite length file.

d. MERGE MODE:

1. COBOL mode selection statement:

(a) SELECT Clause:
SELECT file-ID ASSIGN TO MERGE.

(b) SD Entry. There must be an SD for file-ID. Since the key compare routine is generated using the record description of the SD entry, all files to be merged must have similar record descriptions insofar as key locations and length are concerned.

(c) Merge Verb Syntax. MERGE file-name-1 ON KEY (syntax is identical to KEY syntax for SORT statement)

USING file-name-2 [file-name-3] ... , file-name-9
{ GIVING file-name-10 / OUTPUT PROCEDURE IS procedure-name [THRU procedure-name] }

2. Input file specifications:
   (a) Up to eight files can be merged simultaneously.
   (b) The input files to be merged must have fixed-length records.
   (c) Mixing files, some blocked and some unblocked, is allowable.
   (d) All files must have two and only two buffers (one alternate). This is checked at compile time and will cause a "merge syntax error" if other than two buffers are specified.
   (e) Mixing files as to type is allowed; that is, files from card readers, tapes, and disk can be merged simultaneously.
   (f) A file specified in the merge may be declared optional. If the file is not present when requested, it will be excluded from the merging process, in response to the optional file message.
   (g) If the files to be merged are on magnetic tape, they may be multi-reel files.
   (h) If a file to be merged is not in sequence, an out of sequence condition will occur in the output. An output procedure may be used to detect an out of sequence output from the merge. An out of sequence condition will not cause the merge to be terminated.

3. Output specifications.
   (a) The output media may be a file or a procedure.
   (b) Any specification or restriction specified for sort output procedures is true for merge output procedures.
   (c) Any specifications or restrictions specified for sort output files is true for merge output files.

Refer to the OBJECT-COMPUTER clause for specifications of the core and disk size.

STOP

The STOP statement causes termination or temporary suspension of an object program.

The STOP statement has two options as follows:

Option 1:

```
STOP  RUN
```

Option 2:

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

When the STOP RUN is executed, all OPEN files are closed, all storage areas are released, and the task is removed from the mix. STOP RUN is not used for temporary stops within a program. STOP RUN must be the last statement of its sequence. Use of STOP literal-1 or STOP identifier-1 causes the list of operands to be displayed and the object program to be temporarily suspended. The program must be activated by an "OK" message from the console.

When STOP RUN is executed in a procedure compiled at level three or higher, the procedure will exit the outermost block of its host. If the procedure has been processed or called as a co-routine, it will be DS-ed for doing a bad GO TO. If it has been called as a procedure, the outermost block of the stack is exited and an EOT occurs immediately.

If the task which will execute the STOP RUN statement has executed a DETACH on a control-point item or has set the STATUS of a control-point item to -1, then the STATUS of the control-point item must be tested repetitively until STATUS is equal to -1.

SUBTRACT

The SUBTRACT statement is used to subtract one, or the sum of two or more, numeric data items from one or more items, and to set the values of one or more items equal to the results.

The SUBTRACT statement format includes three options, which are as follows:

Option 1:

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

  identifier-m [ROUNDED] [, identifier-n [ROUNDED]] ...
 [; ON SIZE ERROR statement [ELSE statement]]
```

Option 2:

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

 { identifier-m } GIVING
 { literal-m    }

  identifier-n [ROUNDED] [, identifier-o [ROUNDED] ] ...
 [; ON SIZE ERROR statement [ELSE statement]]
```

Option 3:

```
SUBTRACT { CORRESPONDING } identifier-1
         { CORR          }

  FROM identifier-2 [ROUNDED]
 [ ; ON SIZE ERROR statement [ELSE statement]]
```

Each identifier must refer to a numeric elementary item except in option 2, where any identifiers that appear to the right of the word GIVING may refer to data items that contain editing symbols.

The maximum size of each operand is 23 decimal digits. The composite of operands, which is that data item resulting from the superimposition of all operands, excluding the data items that follow the word GIVING, aligned on their decimal points, must not contain more than 23 digits.

In option 1, all literals or identifiers preceding the word FROM are added together and this total is subtracted from identifier-m, identifier-n, etc.; the difference is then stored as the new value of identifier-m, identifier-n, etc.

In option 2, all literals or identifiers preceding the word FROM are added together, subtracted from literal-m or identifier-m, and the result of the subtraction is stored as the new value of identifier-n, identifier-o, etc.

If option 3 is used, identifier-1 and identifier-2 must refer to group items. Corresponding numeric elementary items from the group referenced by identifier-1 are subtracted from, and the result is stored in, numeric elementary items from the group referenced by identifier-2.

For a discussion of the ROUNDED, ON SIZE ERROR, and CORRESPONDING options, refer to the STATEMENT OPTIONS heading in this section.

Either one of the statements shown in the ON SIZE ERROR clause may be conditional or imperative, or the reserved words NEXT SENTENCE may be used instead.

Arithmetic faults will cause program termination unless SIZE ERROR clause is used.

UNLOCK

The UNLOCK statement is used in an asynchronous processing environment in conjunction with the LOCK clause.

The format for the UNLOCK statement is as follows:

```
UNLOCK { identifier       }...
       { lock-identifier  }
       { event-identifier }
```

Identifier must be a COMPUTATIONAL or COMP-1 item; if COMPUTATIONAL, the identifier must be synchronized to a word boundary.

The UNLOCK statement acts as the logical opposite of the LOCK statement by releasing the imposed common resource restriction. Refer to the description of LOCK statement.

The LIBERATE intrinsic of the MCP is invoked by the UNLOCK statement.

USE

The USE statement is employed in certain declaratives and has three different functions:

a. It can specify supplemental procedures for I/O error and label-handling.
b. It can specify procedures to be employed in a parallel processing environment, and
c. It can specify interrupt procedures.

The format for the USE statement has five options which are as follows:

Option 1:

```
USE AFTER { STANDARD ERROR PROCEDURE }
          { RECORD SIZE ERROR       } ON

  { INPUT-OUTPUT                    }
  { I-O                             }
  { INPUT                           }.
  { file-name-1 [,file-name-2]...   }
```

Option 2:

```
USE { BEFORE } STANDARD { BEGINNING } { REEL }
    { AFTER  }          { ENDING    } { FILE }

LABEL PROCEDURE ON { INPUT-OUTPUT                  }
                   { I-O                           }
                   { INPUT                         }.
                   { OUTPUT                        }
                   { file-name-1 [,file-name-2]    }
```

Option 3:

```
USE { EXTERNAL { identifier-1   } AS PROCEDURE }
    { AS GLOBAL { mnemonic-name } PROCEDURE     }

  [ ; WITH { file-name       }    [, { file-name       }] ...
           { local-storage-name }     { local-storage-name }

    ; USING identifier-2 [, identifier-3] ... ] .
```

Option 4:

```
USE AS INTERRUPT PROCEDURE.
```

Option 5:

```
USE AFTER STANDARD { EXCEPTION } PROCEDURE ON { file-name-1 [, file-name-2]... }
                   { ERROR     }              { INPUT                         }
                                              { OUTPUT                        }.
                                              { I-O                           }
                                              { EXTEND                        }
```

USE

A USE statement, when present, must immediately follow a section header in the Declaratives Section. The remainder of the section must consist of zero, one or more procedural paragraphs that define the procedures to be used.

For Example:

```
section-name SECTION.   USE   statement.
(paragraph-name . (sentence) ... ) ...
```

The USE statement itself is never executed; rather, it defines the conditions calling for the execution of the USE procedures. Only CALL, EXECUTE, PROCESS or RUN statements may reference section-name. There must be no reference to the "main body" of the PROCEDURE DIVISION from within a USE section.

Within a given format, a file-name must not be referred to, implicitly or explicitly, in two or more identical USE statements. The same file-name can appear in a different option of the USE statement. However, appearance of a file-name in a USE statement must not cause the simultaneous request for execution of more than one USE declarative. A file-name may not represent a sort-file. Within a USE procedure, there must not be any reference to any non-declarative procedures. Conversely, in the non-declarative portion there must be no reference (other than by a CALL, EXECUTE, PROCESS or RUN) to procedure-names that appear in the declarative portion. A GO TO or PERFORM statement in an Option 1 or Option 2 declarative may reference a paragraph-name in any other Option 1 or Option 2 declarative section.

The designated procedures are executed by the input-output system at the appropriate time as follows:

a. In option 1, after completing the standard input-output error routine on files assigned to DISK, TAPE, DISKPACK, or REMOTE or after making available any record from TAPE of a BLOCK which is less than the specified block size.
b. In option 2, before or after a beginning or ending input label-check procedure is executed.
c. Before a beginning or ending output label is created.
d. After a beginning or ending output label is created, but before it is written.
e. Before or after a beginning or ending input-output label-check procedure is executed.

When option 1 is used for RECORD SIZE ERROR:

a. The RECORD SIZE ERROR is applicable to input TAPE files only.
b. The RECORD SIZE ERROR may not be specified if the file is declared to contain variable length records or blocks.
c. The ON phrase must specify file-name-1, file-name-2...instead of INPUT.

When option 2 is used:

a. If the file-name option is used, the file description entry for file-name must not specify LABEL RECORDS ARE OMITTED.
b. If the INPUT, OUTPUT, or I-O clause is used, the procedures will not be executed for any INPUT, OUTPUT, or I-O file whose file description entry specifies LABEL RECORDS ARE OMITTED.
c. If BEGINNING or ENDING is not included, the designated procedure will be executed for both beginning and ending labels.
d. If REEL or FILE is not included, the designated procedure will be executed for the appropriate REEL and FILE labels.
e. Option 2 is applicable only to files assigned to tape.

Within the procedures in a USE declarative in which the USE sentence contains either the INPUT or the OUTPUT option, references to label items must be qualified.

Option 3 enables untyped procedures or subroutines to be declared GLOBAL in the same way as they are declared EXTERNAL. This statement is placed in the header of a section to be used as a task.

If the EXTERNAL phrase is used, it identifies the separately compiled program which is to be used as the task when this section is referenced. In addition, there must be no paragraphs in this section when this phrase is employed. Identifier-1 must be defined in the WORKING-STORAGE SECTION such that its value may be a program-name. If the mnemonic-name is used, it must be defined in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION.

Local-storage-name must be defined in the LOCAL-STORAGE SECTION and must be unique among USE statements. A local-storage-name must be present if the USING phrase is present.

If the EXTERNAL phrase is used, the USING phrase is included in the USE statement if, and only if, there is a USING phrase in the PROCEDURE DIVISION header of the referenced, separately compiled program. The number, type and order of the operands in the two USING phrases must then be identical.

When GLOBAL is specified, an untyped procedure must exist in the host with the same name as the section to which the USE is attached.

Identifier-2, identifier-3, ..., must be uniquely defined as files in the FILE SECTION or as level 01 or 77 items in the LOCAL-STORAGE associated with the USE procedure. They may describe any combination of data items, control-point items, event items, index data items, or lock items.

Option 4 of the USE statement is used to specify a declarative as an interrupt procedure.

An interrupt procedure provides a means of interrupting a process when an event-item attached to that procedure is caused. The USE statement may thus be used to declare such interrupt procedures.

Statements which are to be executed when the event is caused and the interrupt procedure is allowed, must follow the USE statement.

Option 5 of the USE statement is used as part of the ANSI 74 Sequential I/O Module specification; thus, requiring that the ANSI74 system dollar option be set.

The words EXCEPTION and ERROR are synonymous.

Specifically, this USE routine option will function the same as Option 1 with the following exception: READ statements may have optional AT END clauses's. This means if a READ statement which contains no AT END clause gets an end-of-file exception, the appropriate USE routine will be executed. If no USE routine exists, the program will be terminated.

WAIT

The WAIT or AWAIT statement is used as part of the user's control of direct access input-output operations and for communication between processes in an asynchronous processing environment.

The format of the WAIT statement is as follows:

Option 1:

```
WAIT  { formula          }
      { event-identifier }
      { INTERRUPT        }
```

Option 2:

```
WAIT control-point-identifier ([subscript,] EXCEPTIONEVENT)
```

Option 3:

```
WAIT [AND RESET]  [formula]
     event-identifier-1 [, event-identifier-2]...
     [GIVING data-name-1]
```

Option 4:

```
WAIT area-identifier [ON EXCEPTION statement [ELSE statement]]
```

All parentheses are required. When a subscript is specified, a maximum of one subscript or index is allowed and it must be followed by a comma.

Option 1 of the WAIT statement containing an event-identifier will cause processing to be suspended until event-identifier has been caused explicitly by a CAUSE statement, or implicitly by option 3 of the READ statement or option 4 of the WRITE statement. The EVENT is not automatically reset.

If formula is used in option 1, the program will be suspended for the number of seconds (or fraction thereof) specified. Formula must be greater than or equal to zero. A control-point-attribute may be specified as part of formula. However, the formula must then by enclosed in parenthesis and the specified attribute must be typed INTEGER or typed REAL.

The INTERRUPT phrase causes execution of this task to be suspended until at least one of its interrupt procedures has been executed.

Option 2 is identical to an option 1 with an event-identifier specified. The event-identifier in this case is the EXCEPTIONEVENT task attribute associated with the control-point-identifier.

Option 3 causes a WAIT for the number of seconds specified by formula (if specified) or until one of the event-identifiers has been CAUSE'd. If the AND RESET phrase was specified, the event-identifier which caused the WAIT to be terminated will be RESET. If the number of seconds specified by "formula" elapse before one of the event-identifiers is CAUSE'd, the AND RESET has no effect. The GIVING phrase provides an integer value to indicate by which method the WAIT was terminated. Thus, the data-name will contain a 1 if the wait was terminated because the number of seconds specified in "formula" had elapsed, a 2 if event-identifier-1 was CAUSE'd, a 3 if event-identifier-2 was CAUSE'd, etc. If a "formula" was not specified, then data-name will contain a 1 if the WAIT was terminated because event-identifier-1 was CAUSE'd, a 2 if event-identifier-2 was CAUSE'd, etc.

Option 4 of the WAIT statement will cause processing to be suspended until the DIRECT input-output operation (option 3 of the READ statement or option 5 of the WRITE) is complete. Area-name is the name of an array specified as a direct array by use of the RECORD AREA clause.

The ON EXCEPTION phrase of option 4 offers a means of detecting abnormal file conditions when DIRECT I/O operations are used. The specific condition may be determined by using the attributes described in the B 7000/B 6000 INPUT/OUTPUT SUBSYSTEM REFERENCE MANUAL, Form No. 5001779.

The following are examples of valid WAIT statements:

```
WAIT CP-I(5,EXCEPTIONEVENT).
WAIT MYSELF(EXCEPTIONEVENT).
WAIT EVENT-NAME.
WAIT X DIV 2.
WAIT (X).
WAIT CP-I(3,TASKVALUE).
WAIT (MYSELF(TASKVALUE)).
WAIT INTERRUPT.
WAIT 35 EVENT-1, EVENT-2, EVENT-3, GIVING XYZ.
WAIT AND RESET 10 EVENT-6.
WAIT BUF-NAME ON EXCEPTION STOP "ERROR".
```

WRITE

The WRITE statement is used to release a logical record for an output file. It can also be used for vertical positioning of a printer. For disk files, the WRITE statement also allows the performance of a specified statement if the contents of the associated actual key data item are found to be invalid.

The WRITE statement has four options which are as follows:

Option 1:

```
WRITE  record-name [FROM identifier-1]

       [ { BEFORE } ADVANCING TO { CHANNEL integer-1          } ]
       [ { AFTER  }              { integer-2 LINES            } ]
                                 { identifier-2 LINES         }
                                 { mnemonic-name              }
                                 { PAGE                       }

       [ ; AT { END-OF-PAGE } statement [ELSE statement ] ]
              { EOP         }
```

Option 2:

```
WRITE  record-name [FROM identifier-1 ]

         ; INVALID KEY statement [ELSE statement]
```

Option 3:

```
WRITE record-name [FROM identifier-1]  TO { AUXILIARY }
                                          { ALTERNATE }
                                          { ERROR     }
```

Option 4:

```
WRITE file-name [KEY IS formula]

        FROM identifier  [USING event-identifier]

        [ ; ON EXCEPTION statement [ELSE statement ] ]
```

As in the READ statement, the file must have been opened prior to any reference to records in it by a WRITE statement. Record-name must be defined in the DATA DIVISION by means of an 01 level entry under the FD entry for the file.

When the WRITE statement is executed at run 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 automatically by the B 7000/B 6000 I/O routine and written by that routine when the block is of the proper length. This information is specified in the FILE clause entries of the DATA DIVISION.

If the FROM clause is used, the operation is identical to a MOVE followed by a WRITE without the FROM clause. The information in the record-name area is no longer available; the information in identifier-1 is unchanged. It is illegal for record-name and identifier-1 to be the same name.

Both the ADVANCING phrase and the END-OF-PAGE phrase allow control of the vertical positioning of each record on a representation of a printed page. If the ADVANCING phrase is not used, automatic advancing is provided to cause single spacing after writing (i.e., BEFORE ADVANCING 1 LINES). If the ADVANCING phrase is used, automatic advancing is overridden and spacing is provided as follows:

a. If identifier-2 or integer-2 is specified, the printer page is advanced the number of lines equal to the current value associated with identifier-2 or integer-2. Identifier-2 must be the name of an unsigned integer data item.
b. If PAGE is specified, the record is printed before or after (depending on the phrase used) advancing to the top of the next page (i.e., channel 1) and if LINAGE was specified for the file, then LINAGE-COUNTER will be reset to one.
c. When CHANNEL is used, the MCP will advance the printer to the carriage control channel specified by integer. Only channels one thru 11, inclusive, are valid. If CHANNEL is used with the LINAGE clause, the results may be unpredictable.
d. If the BEFORE phrase is used, the record is printed before the printer page is advanced. If the AFTER phrase is used, the record is printed after the printer page is advanced.

e. If mnemonic-name is specified, the printer will be advanced to the carriage control channel declared for mnemonic-name in the SPECIAL-NAMES paragraph.
f. The WRITE BEFORE ADVANCING clause is more efficient than the WRITE AFTER ADVANCING clause.
g. If END-OF-PAGE is used the LINAGE clause must be specified for the associated file or the PAGESIZE attribute must be set to a value greater than zero.

If the logical end of the printer page is reached during the execution of an option 1 WRITE statement with the END-OF-PAGE phrase, the statement specified in the END-OF-PAGE clause is executed. The END-OF-PAGE limit is specified in the LINAGE clause in the FILE SECTION of the DATA DIVISION or in the PAGESIZE attribute.

If a CLOSE statement for a file has been executed any attempt to WRITE on the file will result in an error termination.

Option 2 must be used to write on files assigned to the DISK. The operation is as follows:

a. 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 a further attempt is made to WRITE into the file. The last segment of a file is specified in the FILE LIMITS clause or in the ASSIGN clause of the ENVIRONMENT DIVISION. 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. In either case, no writing onto DISK will take place, and the information in the record area of memory will still be available.
b. Records will be written onto DISK in either sequential or random mode according to the rules given under ACCESS MODE.
c. For random DISK files, the WRITE statement performs the function of the SEEK statement, unless an explicit SEEK statement for the record specified by the current value of the ACTUAL KEY has been executed prior to the WRITE statement.
d. If the value of the ACTUAL-KEY is changed after a SEEK statement has been given and prior to the WRITE statement, the WRITE statement will access the record selected by the new value of ACTUAL KEY.

Option 3 is used to control stacker selection on a file assigned to a card punch.

Option 4 of the WRITE statement is used with DIRECT I-O. Refer to the discussion of option 3 of the READ statement and figure 7-14 for an explanation of the action taken by the system for DIRECT I-O.

While the ANSI74 system dollar option is set, ELSE phrases will be paired only with IF statements. In particular, ELSE phrases will not be paired with AT END-OF-PAGE, AT EOP, INVALID KEY, and ON EXCEPTION clauses while the ANSI74 system dollar option is set.

Figure 7-17 illustrates the manner in which the PROCEDURE DIVISION is coded.

Burroughs COBOL CODING FORM

PROGRAM	PROCEDURE DIVISION Coding	REQUESTED BY	PAGE 1 OF 1
PROGRAMMER		DATE	IDENT. 73 80

PAGE NO. 1 3	LINE NO. 4 6	7	A 8 11	B 12 72
	01		PROCEDURE DIVISION.	
	02		DECLARATIVES.	
	03		INPUT-RECORD-COUNT SECTION.	
	04		USE BEFORE ENDING FILE LABEL PROCEDURE ON PERSONNEL-MASTER.	
	05		INPUT-TOTAL. MOVE RECORD-COUNT OF PERSONNEL-MASTER TO CTR-1.	
	06		END DECLARATIVES.	
	07		SORTING SECTION.	
	08		SORTER. SORT EXTRACT-FILE ON ASCENDING KEY DIVISION-CODE	
	09		OF SORT-REC DESCENDING KEY JOB-CODE OF SORT-REC,	
	10		EMPL-NUMBER OF SORT-REC	
	11		INPUT PROCEDURE SELECT-RECORDS GIVING SORTED-EXTRACT.	
	12		END-OF-SORT. STOP RUN.	
	13		SELECT-RECORDS SECTION.	
	14		OPEN-THEM-UP. OPEN INPUT PERSONNEL-MASTER.	
	15		GET-RECORDS. READ PERSONNEL-MASTER AT END GO TO SHUT-UP.	
	16		IF DIV-CODE(INDEX-1, INDEX-2) LESS THAN 19823	
	17		MOVE CORR EMP-REC TO SORT-REC	
	18		RELEASE SORT-REC.	
	19		GO TO GET-RECORDS.	
	20		SHUT-UP. CLOSE PERSONNEL-MASTER WITH RELEASE.	

Figure 7-17. Procedure Division Coding

8. THE COBOL LIBRARY

The COBOL library contains text that is available to a source program at compile time. The effect of the compilation of the library text is the same as if the text were actually written as part of the source program.

The COBOL library may contain text available thru the use of a $FROM or a COPY statement for the ENVIRONMENT DIVISION, the DATA DIVISION, and the PROCEDURE DIVISION.

The COBOL library may be ASCII or EBCDIC coded (14-word records) or BCL coded (10-word records). (The compiler will determine the blocking factor of the file from the label.)

The format for the COPY statement is as follows:

```
COPY  library-name [FROM integer-1] [{THRU    } integer-2]
                                     {THROUGH }

                   [REPLACING word-1 BY text-1 [,word-2 BY text-2] ... ].
```

Word-1, word-2 ... may be any word in a library routine conforming to the rules of definition for words. Word-1, word-2 ... must not be literals.

The COPY statement may appear as follows:

a. Following any paragraph header in the ENVIRONMENT DIVISION. The library entry may, but is not required to, start with the paragraph header.
b. Following the file-name specified for an FD or an SD. If the first library entry processed is an FD or an SD, then the "FD" or "SD" and the following word (presumably, a dummy file-name) from the library, is ignored by the compiler.
c. Following the report-name specified for an RD. If the first library entry processed is an RD, then the "RD" and the following word (presumably, a dummy report-name) from the library, is ignored by the compiler.

d. Following the data-name specified for any data description entry. If the first library entry processed has the same level number as was associated with the COPY statement, the level number and the following word (presumably, a dummy data-name) from the library, are ignored by the compiler.

e. Following a section header. The first library entry processed must be a paragraph-name.

f. As any sentence of a paragraph.

COPY must be a sentence by itself; that is, the COPY statement must end with a period. No other statement or clause may appear on the same source image as a COPY statement. The last source image brought into a program by means of a COPY statement must end in a period.

Library-name may take one of two forms:

a. It may appear as a unique identifier. The identifier becomes the internal name of the compiler's library file and may then be label-equated to the desired file. For example, the COPY statements

```
COPY TAXES.
COPY PAYCALC.
```

would require the following label equation cards:

```
COBOL FILE TAXES (TITLE=TAXES)
COBOL FILE PAYCALC (TITLE=LIB/PAYROLL)
```

b. It may appear as a non-numeric literal. The non-numeric literal becomes the actual external title of the desired file. No label equation for the attribute TITLE is possible since the non-numeric literal specified in the COPY statement is used to dynamically set TITLE. Label equation of other attributes must specify the compilers internal name for the library (i.e.: LIBRARY).

In either case the library may be a BCL file with 10 word records or an EBCDIC file with 14 word records. The file may be blocked or unblocked. The attribute KIND is assumed to be DISK but may be label equated to other devices. For example, if the library to be accessed by the statement COPY "PAYROLL/PRIME/MASTER" is on a diskpack named PAYLIB, the following label equation would be required:

```
COBOL FILE LIBRARY (KIND=PACK, PACKNAME=PAYLIB)
```

If the FROM phrase is specified, copying will start at the sequence number specified. If the THRU phrase is specified, copying will continue until that sequence number has been copied.

The library text is copied from the library, and the result of the compilation is the same as if the text were actually a part of the source program.

If the REPLACING phrase is used, each word from the library text, specified in the format, is functionally replaced by the respective text from the BY phrase, even though it is not shown on the listing.

Text-1, text-2..., may be any desired combination of COBOL characters with two exceptions:

a. The word BY will be taken as the BY of another word-text replacement.
b. The character period, outside of a literal, will be taken as the terminal period for the COPY sentence.

Use of the REPLACING phrase does not alter the material as it appears on the program listing.

At compilation time, the COPY statement and the library text both appear on the output listing. The library text is flagged with the letter L on the listing.

The text contained on the library must not contain any COPY statements.

Example:

PAYROLL/DESC is the library-name of a file containing the file and record descriptions of another file. If the VALUE OF ID clause in its file description is:

```
VALUE OF ID IS PAY-ID
```

and it is desired to give the file an identification of "PAYROLL"/"FILE1", then the following statement may be written in the file section.

```
FD  PAY-MASTER COPY "PAYROLL/DESC"
    REPLACING PAY-ID BY "PAYROLL"/"FILE1".
```

9. ATTRIBUTES

This section is not concerned with the attributes themselves, but only with the way in which attributes are used in COBOL. For a detailed discussion of each attribute, its values, and the mnemonics associated with them, refer to the B 7000/B 6000 INPUT/OUTPUT SUBSYSTEM REFERENCE MANUAL, Form No. 5001779.

There are two types of attributes: physical and logical. Most physical attributes, such as file-name (TITLE), can be label-equated. Logical attributes, such as file-name (PRESENT), cannot be label-equated.

Physical attributes are of two kinds: fixed and variable. A fixed attribute will not change, and cannot be changed, while the file is open. A variable file attribute can change, and may be changed, while the file is open.

Physical attributes can be declared at three different processing points:

a. All physical attributes are specified by the information in the FILE and RECORD descriptions, the ENVIRONMENT DIVISION, the OPEN statements, etc. Both implicit and explicit actions are involved, since many attributes have default values; e.g., tape density by default is "unit selected". Peripheral type mnemonics are explicitly specified by the hardware-name in the ASSIGN clause.
b. Label equation cards may be used at compile time or execute time to specify physical attributes.
c. During the execution of the program, attributes may be interrogated or set by PROCEDURE DIVISION statements.

File and buffer attributes permit dynamic file definition. This means that the precise nature of all characteristics of a file need not be known when a program is written.

Task attributes permit access to and control of the status of a task.

ATTRIBUTE-IDENTIFIER

The attribute-identifier must have the following format:

```
(([subscript,] attribute-name [, formula-1])
```

The subscript may be specified when using a direct switch-file and then specifies which file in the switch-file list is to be addressed. A subscript may also be specified when an attribute of a control-point-identifier (task attribute) is to be accessed and a subscript is required due to descriptive clauses. A maximum of one subscript is permitted.

The attribute-name is treated as a reserved word only when used within an attribute-identifier. An attribute-name may be used as a data-name, procedure-name, etc., unless it is also a reserved word when used outside of an attribute-identifier.

Formula-1 is used when addressing the station attributes to specify the relative station number of the data communications file which is to be addressed. Formula-1 is also used for accessing an attribute which requires a parameter (e.g., AREACLASS).

FILE AND BUFFER ATTRIBUTES

Setting File and Buffer Attributes

B 7000/B 6000 COBOL uses a variation of the SET statement to change file or buffer attributes. The setting of file or buffer attributes is in no way connected with the setting of indexes. For this reason, the syntax for setting file or buffer attributes is included here rather than in the description of the SET statement.

```
SET  { buffer-name }
     { file-name   } (attribute-identifier)  { TO       }  { identifier-1                  }
     { FILE        }                         { UP BY    }  { literal-1                     }
                                             { DOWN BY  }  { formula-2                     }
                                                           { VALUE attribute-mnemonic      }
```

File-name "qualifies" the attribute-name enclosed in the required parentheses. Within a USE procedure, the reserved word FILE can be used, instead of file-name-1, and will refer to the file which has currently invoked the USE procedure. PERFORM statements should not reference a paragraph which makes use of FILE unless FILE has a current meaning to that paragraph.

The file attribute FAMILY is the only attribute that may be SET UP BY or SET DOWN BY. FAMILY can be used only for data communications files.

The choice of literal-1, formula-2, identifier-1, or the attribute mnemonic depends on the type of attribute and the static or dynamic nature of the value to which the attribute is to be set.

Attributes of type POINTER accept or return an alphanumeric item. For example, consider:

a. SET FYLEX (TITLE) TO "A/B/C."

b. MOVE "A/B/C." TO EXT-NAME.
SET FYLEX (TITLE) TO EXT-NAME.

c. SET FORMFIL (FORMMESSAGE) TO "SET DEVICE NOW."

Example a. or b. above would change the external name of FYLEX to A/B/C. The contents of EXT-NAME must be one or more names, each separated by slashes, with a period (.) immediately following the last or only name.

All other attributes return or accept a numeric value of identifier-1, formula-2, literal-1, or the attribute-mnemonic. If the value is not within the range of the specified attribute, an error will occur either at compile time (in the form of a compilation error) or at execute time.

The attribute-mnemonics are not treated as COBOL reserved words. They are "reserved" only within the context in which they are used. The file-name and parentheses define the context for attribute-names, and the COBOL reserved word VALUE defines the context of the attribute mnemonic names. Attribute names and their mnemonic names may be used as data-names or procedure-names in the program if they are not COBOL reserved words.

Examples:

```
SET BUFF-NAME (IOMASK) TO 640.
SET REMOTE-FILE (FAMILY) DOWN BY "NDLNAME1".
SET DISK-FILE (FLEXIBLE) TO VALUE TRUE.
SET TAPE-OUT(SAVEFACTOR) TO 180.
SET FILE-OUT (DATE) TO CC-AS-OF-DATE.
SET SW-FILE(3,AREACLASS,28) TO 9.
```

Interrogating File and Buffer Attributes

File or buffer attributes can also be interrogated by appearing as the sending field in a MOVE statement or as the subject or object of a condition. In addition, attributes whose class is implicitly numeric may be used in DISPLAY statements and in place of any identifier in an arithmetic statement except the receiving field identifier. When an attribute is moved into an area by a MOVE statement, the receiving field must be an elementary numeric item described as numeric except for the type POINTER attributes, where the receiving

field must be alphanumeric. BOOLEAN attributes (those attributes having mnemonic values of TRUE or FALSE) will return a zero if FALSE and a "1" if TRUE. REAL or INTEGER attributes should be moved or compared to a COMP or COMP-1 receiving field. REAL attributes which will not have sign bits or exponent bits set may be associated with an integer of any USAGE. The file attribute STATE should be moved to an integer COMP or COMP-1 receiving field and then manipulated with option 4 MOVE statements.

A file or buffer attribute appearing in a condition may be tested against its associated attribute mnemonic.

Examples:

```
IF FILE-IN (KIND) = VALUE PRINTER
MOVE TAPE-IN (RECORD) TO RECORD-CNT.
IF TAPE-IN (TITLE) IS EQUAL TO "A/FILE/NAME." ...
PERFORM READ-A-REC THRU RR-XIT UNTIL FILE-IN
(EOF) = VALUE TRUE.
MOVE REMOTE-FILE (TITLE) TO HOLD-NDL-NAME.
```

TASK ATTRIBUTES

Task attributes are used to change or interrogate the task variables of related processes in a synchronous or asynchronous processing environment. This document is concerned only with the way in which task attributes are referenced in B 7000/B 6000 COBOL. The user should be familiar with the concepts of tasking, the task attributes, and their possible variations.

Setting Task Attributes

B 7000/B 6000 COBOL uses a variation of the SET statement to change task attributes. The setting of task attributes is in no way connected with the setting of indexes. For this reason, the syntax for setting task attributes is included here rather than in the description of the SET statement in Chapter 7.

```
SET { MYJOB                    } (attribute-identifier)
    { MYSELF                   }
    { control-point-identifier }

    TO { identifier-1                  }
       { literal-1                     }
       { formula-1                     }
       { VALUE attribute-mnemonic      }
```

Control-point-identifier may be attached to a program. The reserved word MYSELF is a compiler supplied control-point item which refers to the task within which it appears. Thus, any attribute of a given task may be referenced within that task as MYSELF (attribute-identifier). The reserved word MYJOB is a compiler supplied control-point item which refers to the job within which the task appears. Thus, any attribute of a job may be referenced within any task of that job as MYJOB (attribute-identifier).

The choice of identifier, literal, formula-2, or mnemonic-name depends on the attribute being set and its desired value. The type POINTER task attributes accept or return either an alphanumeric item.

Examples:

a. SET TSK (1,NAME) TO "A/B/C.".
b. MOVE "A/B/C." TO EXT-NAME. SET TSK (1,NAME) TO EXT-NAME.

The examples above would change the external name to A/B/C. The contents of EXT-NAME must be one to 14 names (1 to 17 characters in length), each separated by slashes, with a period following the last or only name.

Task attributes which are type EVENT may be used in place of any valid use of an event-identifier (USAGE EVENT).

Task attributes which are type TASK are themselves control-point-identifiers of some other associated task. This type of attribute may be employed to access or manipulate the task attributes of that associated task.

All other task attributes accept or return a numeric value of identifier, formula-2, literal, or the value associated with mnemonic-name. If the value is not within the permissible range for the attribute specified, an error will occur at compile time (syntax error) or at execute time. The attribute-mnemonic is a name associated with a constant value for those attributes that have a set number of predetermined possible conditions.

The attribute names and their mnemonics are not treated as COBOL reserved words. They are "reserved" only within the context in which they are used and may be used as data-names or procedure-names, provided they are not regular reserved words (i.e., the task attribute LOCKED is a reserved word; IOTIME is not).

Following are various examples of the task attribute SET statement:

```
SET PROG1(1,DECLAREDPRIORITY) TO 1.
SET PROG2(1,STATUS) TO -1.
SET PROG3(STACKSIZE) TO 1500.
SET MYSELF (DECLAREDPRIORITY) TO 90.
SET MYSELF(PARTNER(DECLAREDPRIORITY)) TO 65.
```

Interrogating Task Attributes

Task attributes may be interrogated by appearing as the sending field in a MOVE statement or as the subject or object of a condition. In addition, attributes whose class is implicitly numeric may be used in DISPLAY statements and in place of any identifier in an arithmetic statement except the receiving field identifier. When an attribute is moved into an area by a MOVE statement, the usage of the receiving field must be consistent with the usage of the attributes. BOOLEAN attributes (those attributes having mnemonic values of TRUE or FALSE) will return a 0 if FALSE, or a 1 if TRUE. BOOLEAN or INTEGER attributes should be moved to a NUMERIC receiving field. POINTER attributes should be moved to a non-numeric receiving field. REAL attributes should be moved to COMP-4 receiving fields.

Task attributes may be tested against their associated attribute mnemonics. Following are various examples of interrogating task attributes:

```
IF PROG1(1,LOCKED) = VALUE TRUE SET PROG1(1,LOCKED) TO VALUE FALSE.
MOVE PROG3(PROCESSTIME) TO PRINT-P-TIME.
IF PROG2(1,NAME) = "X/Y/Z" ...
PERFORM PRINTING-ROUTINE UNTIL PROG1(2,STATUS) = 3.
```

10. COBOL AND DATA COMMUNICATIONS

This section defines the B 7000/B 6000 COBOL language facilities for handling data communications files. The characteristics of the B 7000/B 6000 data communications system permit B 7000/B 6000 COBOL programs to deal with data communications devices as easily as they deal with other file types. Changes and/or extensions were made to the language only where there were no acceptable constructs present. Language constructs that represent certain characteristics of other file types apply to data communications files where the applications are similar.

The material in this section does not describe the B 7000/B 6000 data communications system. A knowledge of the basic concepts of the data communications system is a prerequisite to the proper use of this material.

An object program can receive data from and send information to remote devices by the use of Datacom files. The Data Communications Subsystem allows essentially the same degree of media independence which can be achieved by an object program dealing with other peripheral files. However, unlike the usual peripheral devices which are accessed through peripheral control units, remote devices are accessed through a special peripheral processor called the Data Communications Processor or DCP.

Datacom files are permanent files in the sense that they are described in the Network Definition and are assigned an external name (TITLE) in the Network Information File. The Network Definition Language allows a Datacom file to be a single station (with the possibility of the external name of the file being the name of the station) or a FAMILY of stations. A FAMILY is a list of stations which are grouped together to form a file in the Network Description.

The file attributes, which are uniquely related to Datacom files, reference information stored and maintained in the station list. There are "Datacom" attributes which deal with the file as a whole, such as FAMILYSIZE, LASTSTATION and CENSUS; and "station" attributes which deal with the characteristics of

the specific stations in the file, such as WIDTH, DISPOSITION, SCREEN and ASSIGNTIME. Some of the more general file attributes, such as TITLE, POPULATION and PAGESIZE, have special meaning when referencing a Datacom file.

When a Datacom file is opened, a station list is created in addition to the File Information Block. The station list contains the information from the Network Information File which is necessary to distinguish the different stations which have been included in the file. When the station list is created, each station in the file is assigned a Relative Station Number in the order in which they are described in the Network Information File. The Relative Station Number is in effect an index into the station list. The Relative Station Number is not unique to the station and the ordering of the station list can be changed by extensive use of the FAMILY attribute. If a station is subtracted from a file, its Relative Station Number becomes invalid and points to an empty area in the station list. A station which is added to a file may be assigned a Relative Station Number which belonged to a station which was subtracted from the file, which is to say that Relative Station Number's are reusable. It is also possible for a valid Relative Station Number to be larger than the value of the FAMILYSIZE attribute after some stations have been subtracted from the file.

An object program may open an unrestricted number of Datacom files, and a Remote station may be assigned to any number of Datacom files. However, a Remote station can be assigned concurrently to only one Datacom file that is input capable without the station's controlling MCS participating in the I/O operations.

An object program can write to a Datacom file in two different ways. If the LASTSTATION attribute is set to a Relative Station Number, the output will be directed to that specific station in the file. If the value of the LASTSTATION attribute is zero (the default value), the output will be broadcast to every station in the file.

An object program receives input from a Datacom file in a first-in-first-out order. After a read statement, the LASTSTATION attribute contains the Relative Station Number of the station from which the input came.

When an object program opens a Datacom file, a FILE OPEN Datacom message is sent for each station in the file to the station"s controlling MCS. The FILE OPEN message is a notification to the MCS that the file would like permission to communicate with the station. The MCS may allow, postpone, or deny assignment of the station to the file.

If the MCS allows assignment, the DCWRITE function checks to see if the requested use of the station is compatible with its description in NDL. If the requested assignment is to an input capable file, DCWRITE checks to make sure that the station is not already assigned to another input capable file, unless the MCS is participating in I/O. If the MCS is not participating in I/O while a station is assigned to an input capable file, DCWRITE will deny requests to assign the station to another file.

If the MCS denies assignment, which it may also do subsequent to having allowed or postponed assignment, an end of file message is placed in the input queue if the file is input capable. Further writes to the station will cause end of file action.

A multi-station Datacom file can have end of file notification for each station in the file. The LASTSTATION attribute can be interrogated to discover which station is no longer assigned to the file after end of file action. The station's DISPOSITION attribute or the qualification field of the STATE attribute can be accessed to find the reason for the end of file.

If an object program opens a Datacom file containing a station which is declared in NDL without line assignment, the controlling MCS of the station is restricted in its response to the FILE OPEN message. Assignment of the station to the file is allowed only if the MCS participates in I/O. MCS participation in I/O requires an agreed discipline between the object program and the MCS, so that the MCS can perform message switching and the object program can identify the source station.

ENVIRONMENT DIVISION CONSIDERATIONS

FILE-CONTROL

Data communications files are selected in the file-control paragraph and assigned to the hardware-name (REMOTE).

The syntax of the SELECT clause for remote files is as follows:

```
SELECT  file-name  ASSIGN TO REMOTE
    [, ACCESS MODE IS SEQUENTIAL]
    [, ACTUAL KEY IS data-name]
```

The ACTUAL KEY is analogous to the ACTUAL KEY of a disk file. Files declaring ACTUAL KEYs have the ACTUAL KEY set to the relative station number when a READ is done. WRITE statements for this file are then "KEYED" WRITES, with the value of the ACTUAL KEY passed to the MCP; thus, directing the record to the specified station when Option 1 of a WRITE statement is used or the files LASTSTATION attribute is set to the value of the ACTUAL KEY just before an Option 2 WRITE statement.

I-O-CONTROL

None of the clauses in the I-O-CONTROL paragraph may be applied to data communications files.

DATA DIVISION CONSIDERATIONS

File Descriptions

Data communications files are declared by a file declaration (FD) as are other file types. However, some of the file description clauses are not relevant to data communications files and may not be used. These clauses are as follows:

a. The RECORDING MODE clause.
b. The BLOCK clause. Blocked records cannot be declared for data communications files. The block size is implicitly equal to the maximum record size.
c. The LABEL RECORDS clause. Label records and user header and trailer records cannot be declared. Recognition of files is handled by the data communications controller and the I/O intrinsics.
d. The LINAGE clause.
e. SAVE-FACTOR. Data communications files (or families) are defined on a permanent basis in the Network Information File by the NETWORK DEFINITION LANGUAGE.

File description clauses used with data communications files are as follows:

a. The RECORD CONTAINS clause. Rules for using this clause are the same as those for other file descriptions. This clause may be omitted for fixed-length records.
b. The DATA RECORDS clause. This clause is optional as in all other file descriptions.
c. The VALUE OF clause. The VALUE OF ID option of this clause is required. The VALUE OF data-name option may not be used. The value specified must be the external name (TITLE) declared in the Network Definition.

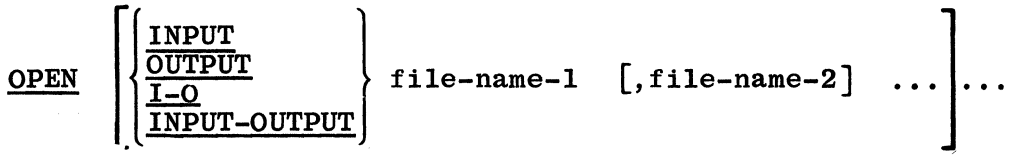
Record Descriptions

The record areas of a data communications file are described like the record areas of other files, although the following special considerations do apply. All data items within the record should be described, either explicitly or implicitly, as DISPLAY, DISPLAY-1, COMP-2 or ASCII. Appropriate translations will be automatically provided. Variable length records may be used just as in any non-datacomm file.

PROCEDURE DIVISION I-O STATEMENTS

OPEN

The syntax is the same as the OPEN statement, described in section 7, except that the optional clauses REVERSED and NO REWIND do not apply to data communications files. The appropriate format is:



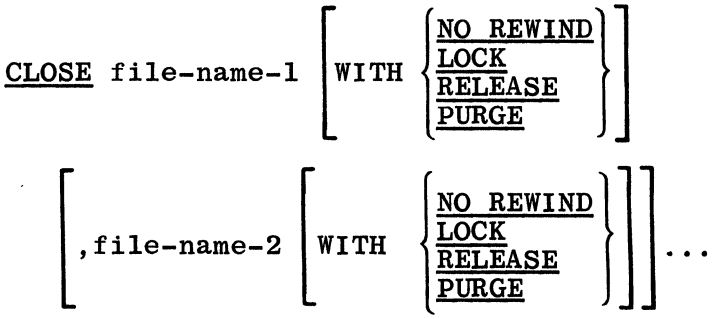
At least one of the following options must be used: INPUT, OUTPUT, or I-O. They may appear in any order.

The OPEN statement causes the allocation of necessary space for alternate areas, file tanks, FIBs, etc.

All stations of the FAMILY associated with the file are assigned a relative station number (RSN); relative station number designation within a FAMILY is determined by the sequence in which stations are specified as members of the FAMILY.

CLOSE

The CLOSE statement returns to the MCP those areas that were allocated to the file.



When LOCK, RELEASE, or PURGE are specified, the files family reverts to that which was declared in the Network Definition. When the WITH clause is not specified or the NO REWIND is specified, changes in the files FAMILY are retained so that a subsequent OPEN will retain the Station List which existed when the file was last closed within the same program.

READ

The READ statement causes the next message in the file to be made available to the object program and sets ACTUAL KEY, if specified, to the relative station number from which the message originated.

```
READ file-name RECORD [INTO identifier-1]
    ;INVALID KEY statement [ELSE statement]
```

See section 7 for a general description of the READ statement.

The abnormal conditions which cause execution of the INVALID KEY clause or the error USE procedures are discussed later in this section.

WRITE

The WRITE statement causes a message to be made available to a file for transmission to one or more stations assigned to that file.

The syntactical formats for WRITE statements to files assigned to remote devices are as follows:

Option 1:

```
WRITE record-name [FROM identifier-1]
    ;INVALID KEY statement [ELSE statement]
```

Option 2:

```
WRITE record-name [FROM identifier-1]
{BEFORE} ADVANCING TO {PAGE                    }
{AFTER }              {CHANNEL integer         }
                      {numeric-operand LINES   }
                      {mnemonic-channel-name   }
```

See section 7 for a general description of the WRITE statement.

A message released to a file is transmitted to the station specified by the relative station number indicated by the contents of the file attribute LASTSTATION. If an ACTUAL KEY has been specified for the file, the LAST-STATION file attribute is replaced by the contents of ACTUAL KEY prior to transmission. If the file attribute LASTSTATION contains the value zero, the available record will be broadcast to all stations assigned to the file.

The abnormal conditions which cause execution of the INVALID KEY clause or the error USE procedures are discussed later in this section.

ABNORMAL CONDITIONS

It is the COBOL programmer's responsibility to be aware of, and to provide for, the various types of abnormal conditions that can occur during the use of data communications files.

The MCP will maintain the information concerning the status of a file and its associated stations. This information is accessible through various attributes.

The END-OF-FILE condition causes the statements specified by the INVALID KEY clause in the READ or WRITE statement to be executed.

SIZE, BREAK, TIMELIMIT and SECURITY errors cause execution of the USE AFTER STANDARD ERROR if specified. If the USE procedure is not specified, the program will be terminated.

Differentiation among the various types of abnormal conditions can be done by interrogating file or station attributes.

The END-OF-FILE condition occurs for the following reasons.

a. For a WRITE statement, when the message is directed to a station which is not currently assigned.
b. For a WRITE statement, when LASTSTATION is zero and no stations are currently assigned to the file.
c. For a READ statement, when all stations in the file have been denied assignment.
d. For a READ statement, subsequent to denial of assignment of each input capable station specified for the file. Subsequent READ statements will not generate an END-OF-FILE condition unless "c" above has occurred or an additional station is denied assignment.

Errors which cause execution of the USE AFTER STANDARD ERROR (or program termination when the appropriate USE procedure is not specified) are:

a. Time Limit. This error occurs when no input is received following a READ or output is not initiated following a WRITE statement in the amount of time specified by the file attribute TIMELIMIT.
b. Size Error. This error is possible only when FILETYPE=1.
c. Security. This error is caused by a security violation.
d. Break. This error occurs at the instigation of an operator at a station. All output queued for that station is discarded. The program will be notified of the "break" action on the next WRITE to that station. (Break is part of the datacom/MCS interface.)

FILE AND STATION ATTRIBUTES

A data communications network has a variety of attributes which may be of interest to object programs. These attributes can be interrogated and/or altered. There are two classes of attributes for data communications files: one set of attributes pertaining to the file; and one set of attributes for each station which is a member of the file. Station-attributes are accessed by specifying their relative station number. In the syntax of the file-attribute SET statement in section 9, formula-1 defines the relative terminal number.

For descriptions of station-attributes and file-attributes pertinent to data communications, refer to the B 7000/B 6000 INPUT/OUTPUT SUBSYSTEM REFERENCE MANUAL, Form No. 5001779, and B 6700/B 7700 DATA COMMUNICATIONS FUNCTIONAL DESCRIPTION, Form No. 5000060.

11. REPORT WRITER

REPORT WRITER is a special-purpose language subset of COBOL which permits a more convenient method of producing reports. REPORT WRITER is non-procedural in nature, in that the major emphasis is placed in the REPORT SECTION of the DATA DIVISION on the logical organization of the report, its format, contents, organization, and structure. The concept of a hierarchy of levels is used in defining this logical organization. Each report is composed of report groups; these, in turn, are divided into sequences of items. PROCEDURE DIVISION code for moving data, constructing print lines, counting line and page numbers, producing heading and footing lines, summing information, and checking for control breaks is generated automatically by the compiler, primarily from the clauses in the REPORT SECTION.

FILE SECTION

REPORT Clause

The REPORT clause specifies the names of reports that comprise a report file.

```
{REPORT IS  }
{REPORTS ARE}   report-name-1   [, report-name-2 ] . . .
```

Each report-name specified in a REPORT clause must be the subject of a Report Description entry in the Report Section. The order of appearance of the report-names is not significant.

A report-name must appear in only one REPORT clause.

The subject of a File Description entry that specifies a REPORT clause may only be referred to by the OPEN statement with the OUTPUT phrase or the CLOSE statement.

The presence of more than one report-name in a REPORT clause indicates that the file contains more than one report.

After execution of an INITIATE statement and before the execution of a TERMINATE statement for the same report file, no other explicit input-output operation may reference that report file or the Record Description entries associated with that report file.

REPORT SECTION

REPORT DESCRIPTION ENTRY

The report description entry contains information pertaining to the overall format and structure of a report named in the FILE SECTION and is uniquely identified in the REPORT SECTION by the level indicator RD.

Option 1:

```
RD  report-name;  COPY library-name

                  [FROM seq. no.] [{THRU    } seq. no.]
                                  {THROUGH  }
                  [REPLACING  word-1  BY  text-1
                          [,  word-2  BY  text-2]  ...].
```

Option 2:

```
RD  report-name

    [;  CODE  literal-1]

    [; {CONTROL IS  } {FINAL                                          }]
       {CONTROLS ARE} {FINAL,  data-name-1  [,data-name-2]...         }
                      {data-name-1 [,data-name-2]...                  }

    [;  PAGE  [LIMIT IS ]  integer-1  [LINE ]
              [LIMITS ARE]            [LINES]

            [,  HEADING  integer-2]
            [,  FIRST DETAIL  integer-3]
            [,  LAST DETAIL  integer-4]
            [,  FOOTING  integer-5]].
```

For information on the COPY option, see Section 8, The COBOL Library.

The level indicator RD identifies the beginning of a report description and must precede the report-name. The report-name must appear in one and only one REPORT clause.

All semicolons are optional in the report description, but the entry must be terminated by a period. The clauses which follow the report-name are all optional, and their order of appearance is immaterial.

Report-name is the highest permissable qualifier that may be specified for LINE-COUNTER, PAGE-COUNTER and all data-names defined within the Report Section.

CODE

If more than one report is associated with a file and the reports are produced simultaneously, the CODE clause must be used so that the individual lines of each report may be identified by the MCP printer backup routine.

The format of the CODE clause is:

CODE literal-1

Literal-1 must be a 2-character non-numeric literal. If CODE is specified for any report in a file, it must be specified for all reports in that file.

When the CODE clause is specified, literal-1 is automatically placed in each record generated. The positions occupied by literal-1 are not included in the description of the print line but are included in the size of a logical record.

The backup printer file on DISK for a report for which the CODE clause is specified will have the title:

BDREPORT/task-number/count id

The task-number is the task number assigned to the task which created the report and is expressed as a seven digit number. The count is 000 for the first time the file is opened within the task, 001 for the second time, etc. The id is taken from the VALUE OF ID specified for the file. For example, if the task mix number is 325, and the VALUE OF ID specifies "ABCD", then the file title assigned to a backup disk file will be:

BDREPORT/0000325/000ABCD

The second time the file is opened within that task, the file title will be:

BDREPORT/0000325/001ABCD

A backup disk file may be printed by entering the following message:

?PB D task-number/count id KEY REPORT EQUAL literal-1

Task-number is the four digit task number assigned by the MCP. Count and id are discussed above. When the KEY clause is used, literal-1 should match the literal-1 used in a CODE clause except that literal-1 when used in the KEY clause of this MCP message must not be bounded by quotes and may not contain blanks or special characters. Example:

?PB D0325/000ABCD KEY REPORT EQUAL A2

A backup tape file of a report writer report for which a CODE clause was specified may be printed by entering the following message:

?PB MT xxx [FILE integer] KEY REPORT EQUAL literal-1

In the above, xxx is the tape unit on which the backup tape resides, and literal-1 is as described for disk files.

CONTROL

```
{CONTROL IS  }  {FINAL                                          }
{CONTROLS ARE}  {data-name-1    [, date-name-2] ...             }
                {FINAL,  data-name-1    [,data-name-2] ...      }
```

FINAL, if used, is the highest control. The positional order in which the data-names appear specifies the hierarchial level from major to minor; data-name-1 is the major control, data-name-2 is the intermediate level control, and the last data-name specified is the minor control. The data-names cannot be subscripted or indexed. The data-names must be defined as group or elementary items in the FILE, WORKING-STORAGE, or LINKAGE sections of the DATA DIVISION. Control data items are subject to the same rules that apply to the SORT key data-names.

The CONTROL clause is required when control heading and/or control footing groups are used. The data-names specified in the CONTROL clause are the only data-names referred to by the RESET and TYPE clauses in the report group descriptions for a report. No data-name, including FINAL, may be referred to by more than one type control heading report group and one type control footing report group.

The CONTROL clause establishes the levels of the control hierarchy for the report.

Data-name-1 and data-name-2 must not be defined in the Report Section. Data-name-1 and data-name-2 may be qualified but must not be subscripted or indexed. Each data-name must identify a different data item. Data-name-1, data-name-2, . . . , must not have subordinate variable-occurrence data-items. Control data items are subject to the same rules that apply to SORT keys.
The data-names and the word FINAL specify the levels of the control hierarchy. FINAL, if specified, is the highest control, data-name-1 is the major control, data-name-2 is an intermediate control, etc. The last data-name specified is the minor control.

The execution of the chronologically first GENERATE statement for a given report causes the values of all control data items associated with that report to be saved. On subsequent executions of all GENERATE statements for that report, control data items are tested for a change of value. A change of value in any control data item causes a control break to occur. The control break is associated with the highest level for which a change of value is noted.

A control break is tested for by comparing the contents of each control data item with the prior contents saved from the execution of the previous GENERATE statement for the same report. The relation test is applied as follows:

a. If the control data item is a numeric data item, the relation test is for the comparison of two numeric operands.
b. If the control data item is an index data item, the relation test is for the comparison of two index data items.
c. If the control data item is a data item other than as described in a. and b. above, the relation test is for the comparison of two non-numeric operands.

A control break for FINAL occurs before the first detail line is printed and when a TERMINATE statement is executed. A control break occurring at a particular level implies a control break for each lower level in the control hierarchy. For example, if this CONTROL clause is used:

```
CONTROLS ARE MAJ-KEY, INT-KEY, MIN-KEY
```

and control headings and footings are specified, they will be printed in the following order, upon a control break on MAJ-KEY:

```
CONTROL FOOTING   (for MIN-KEY)
CONTROL FOOTING   (for INT-KEY)
CONTROL FOOTING   (for MAJ-KEY)
CONTROL HEADING   (for MAJ-KEY)
CONTROL HEADING   (for INT-KEY)
CONTROL HEADING   (for MIN-KEY)
```

PAGE LIMIT

```
PAGE [LIMIT IS ]      integer-1 [LINE ]
     [LIMITS ARE]                [LINES]

[, HEADING integer-2] [, FIRST DETAIL integer-3] [, LAST DETAIL
integer-4] [, FOOTING integer-5]
```

The PAGE LIMIT clause is required when page formatting must be controlled by the REPORT WRITER. The PAGE LIMIT clause may be omitted when no association is desired between report groups and the physical format of an output page. If this clause is omitted, neither a PAGE HEADING nor a PAGE FOOTING can be declared.

The PAGE clause defines the length of a page and the vertical subdivisions within which report groups are presented.

The HEADING, FIRST DETAIL, LAST DETAIL and FOOTING phrases may be written in any order.

Integer-1 must not be greater than 255.

Integer-2 must be greater than or equal to 1 (one).

Integer-3 must be greater than or equal to integer-2.

Integer-4 must be greater than or equal to integer-3.

Integer-5 must be greater than or equal to integer-4.

Integer-1 must be greater than or equal to integer-5.

The following rules indicate the vertical subdivision of the page in which each TYPE of report group may appear when the PAGE clause is specified.

a. A REPORT HEADING report group that is to be presented on a page by itself, if defined, must be defined such that it can be presented in the vertical subdivision of the page that extends from the line number specified by integer-2 to the line number specified by integer-1, inclusive.

   A REPORT HEADING report group that is not to be presented on a page by itself, if defined, must be defined such that it can be presented in the vertical subdivision of the page that extends from the line number specified by integer-2 to the line number specified by integer-3 minus 1, inclusive.

b. A PAGE HEADING report group, if defined, must be defined such that it can be presented in the vertical subdivision of the page that extends from the line number specified by integer-2 to the line number specified by integer-3 minus 1, inclusive.

c. A CONTROL HEADING or DETAIL report group, if defined, must be defined such that it can be presented in the vertical subdivision of the page that extends from the line number specified by integer-3 to the line number specified by integer-4, inclusive.

d. A CONTROL FOOTING report group, if defined, must be defined such that it can be presented in the vertical subdivision of the page that extends from the line number specified by integer-3 to the line number specified by integer-5, inclusive.

e. A PAGE FOOTING report group, if defined, must be defined such that it can be presented in the vertical subdivision of the page that extends from the line number specified by integer-5 plus 1 to the line number specified by integer-1, inclusive.

f. A REPORT FOOTING report group that is to be presented on a page by itself, if defined, must be defined such that it can be presented in the vertical subdivision of the page that extends from the line number specified by integer-2 to the line number specified by integer-1, inclusive.

   A REPORT FOOTING report group that is not to be presented on a page by itself, if defined, must be defined such that it can be presented in the vertical subdivision of the page that extends from the line number specified by integer-5 plus 1 to the line number specified by integer-1, inclusive.

All report groups must be described such that they can be presented on one page. A multi-line report group is never split across page boundaries.

The vertical format of a report page is established using the integer values specified in the PAGE clause.

a. Integer-1 defines the size of a report page by specifying the number of lines available on each page.

b. HEADING integer-2 defines the first line number on which a REPORT HEADING or PAGE HEADING report group may be presented.

c. FIRST DETAIL integer-3 defines the first line number on which a body group may be presented. REPORT HEADING and PAGE HEADING report groups may not be presented on or beyond the line number specified by integer-3.

d. LAST DETAIL integer-4 defines the last line number on which a CONTROL HEADING or DETAIL report group may be presented.

e. FOOTING integer-5 defines the last line number on which a CONTROL FOOTING report group may be presented. PAGE FOOTING and REPORT FOOTING report groups must follow the line number specified by integer-5.

If absolute line spacing is indicated for all report groups, integer-2 thru integer-5 do not need to be specified. If relative line spacing is indicated for individual detail report group entries, some or all of the limits must be defined, depending on the type of report groups within the report, in order for control of page formatting to be maintained.

If the PAGE clause is specified the following implicit values are assumed for any omitted phrases:

a. If the HEADING phrase is omitted, a value of one (1) is assumed for integer-2.

b. If the FIRST DETAIL phrase is omitted, a value equal to integer-2 is given to integer-3.

c. If the LAST DETAIL and the FOOTING phrases are both omitted, the value of integer-1 is given to both integer-4 and integer-5.

d. If the FOOTING phrase is specified and the LAST DETAIL phrase is omitted, the value of integer-5 is given to integer-4.

e. If the LAST DETAIL phrase is specified and the FOOTING phrase is omitted, the value of integer-4 is given to integer-5.

If the PAGE clause is omitted, the report consists of a single page of indefinite length.

Figure 11-1 illustrates page format control of report groups when the PAGE LIMIT clause is specified.

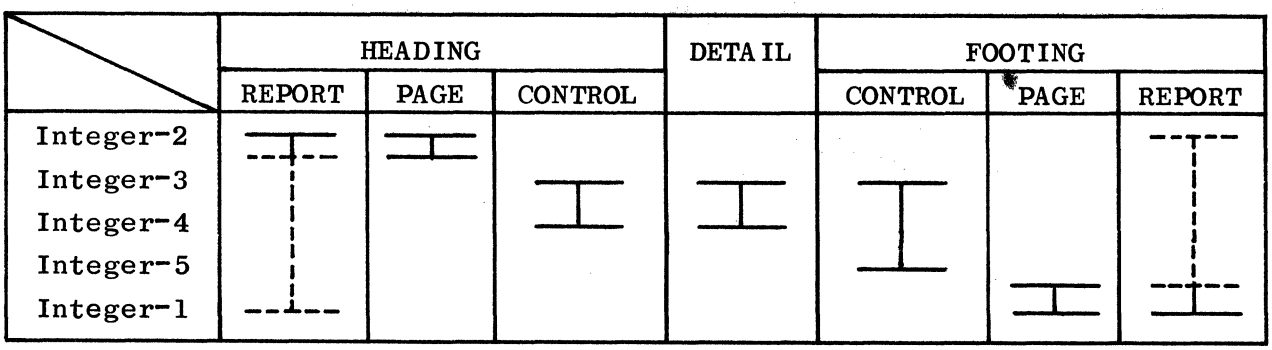


Figure 11-1. Page Format Control

Absolute line number or absolute NEXT GROUP spacing must be consistent with controls specified in the PAGE LIMIT clause.

Page regions that are established by the PAGE clause are described in figure 11-2.

Report Groups That May Be Presented In The Region	First Line Number Of The Region	Last Line Number Of The Region
REPORT HEADING described with NEXT GROUP NEXT PAGE REPORT FOOTING described with LINE integer-1 NEXT PAGE	integer-2	integer-1
REPORT HEADING not described with NEXT GROUP NEXT PAGE PAGE HEADING	integer-2	integer-3 minus 1
CONTROL HEADING DETAIL	integer-2	integer-4
CONTROL FOOTING	integer-3	integer-5
PAGE FOOTING REPORT FOOTING not described with LINE integer-1 NEXT PAGE	integer-5 plus 1	integer-1

Figure 11-2. Page Regions

SPECIAL COUNTERS

Two special counters are automatically supplied for each report described in the REPORT SECTION. The special counters are PAGE-COUNTER and LINE-COUNTER.

PAGE-COUNTER

PAGE-COUNTER is the reserved word used to reference a special register that is automatically created for each report specified in the Report Section.

In the Report Section, a reference to PAGE-COUNTER can only appear in a SOURCE clause. Outside of the Report Section, PAGE-COUNTER may be used in any context in which a data-name of integral value can appear.

If more than one PAGE-COUNTER exists in a program, PAGE-COUNTER must be qualified by a report-name whenever it is referenced in the Procedure Division. In the Report Section, an unqualified reference to PAGE-COUNTER is implicitly qualified by the name of the report in which the reference is made; whenever the PAGE-COUNTER of a different report is referenced, PAGE-COUNTER must be explicitly qualified by that report-name.

Execution of the INITIATE statement causes the PAGE-COUNTER of the referenced report to be reset to the value one (1).

PAGE-COUNTER is automatically incremented by one (1) each time a page advance is executed.

PAGE-COUNTER may be altered by Procedure Division statements. If a starting value other than one (1) is desired, the contents of PAGE-COUNTER should be changed following the INITIATE statement for that report.

LINE-COUNTER

LINE-COUNTER is the reserved word used to reference a special register that is automatically created for each report for which the PAGE LIMIT clause is specified. If more than one report may have a LINE-COUNTER, then all references to LINE-COUNTER must be qualified. In the Report Section, an unqualified reference to LINE-COUNTER is implicitly qualified by the name of the report in which the reference is made.

In the Report Section, a reference to LINE-COUNTER can only appear in a SOURCE clause. Outside of the Report Section, LINE-COUNTER may be used in any context in which a data-name of integral value may appear. Changing the LINE-COUNTER by Procedure Division statements may cause page format control to become unpredictable.

Execution of an INITIATE statement causes the LINE-COUNTER for that report to be reset to zero (0). LINE-COUNTER is also reset to zero (0) each time a page advance is executed for the associated report.

After a report group is printed, LINE-COUNTER contains the line number on which the last line of the report group was printed unless the report group specifies the NEXT GROUP clause in which case LINE-COUNTER will contain zero (0) if NEXT PAGE was specified or the line number specified.

For further information on line number positioning, see the LINE NUMBER and NEXT GROUP clauses.

REPORT GROUP DESCRIPTIONS

Following each report description RD entry are one or more report groups. Each group describes one or more print lines related to a specific function in producing a report. A report group is described by a hierarchial data structure similar to record descriptions in the other sections of the DATA DIVISION.

Option 1:

```
01   data-name-1 ;   COPY   library-name

          [FROM  sequence-number] [{THRU   }  sequence-number]
                                   {THROUGH}

          [REPLACING  word-1  BY  text-1  [, word-2 BY  text-2] ...] .
```

Option 2:

```
01 [data-name-1]

[; LINE NUMBER IS     {integer-1    ON NEXT PAGE}]
                      {PLUS  integer-2          }

[; NEXT GROUP IS      {integer-3      }]
                      {PLUS integer-4 }
                      {NEXT PAGE      }

                    { {REPORT HEADING }                          }
                    { {RH             }                          }
                    {                                            }
                    { {PAGE HEADING   }                          }
                    { {PH             }                          }
                    {                                            }
                    { {CONTROL HEADING}   {data-name-2}          }
                    { {CH             }   {FINAL      }          }
                    {                                            }
 ; TYPE IS          { {DETAIL         }                          }
                    { {DE             }                          }
                    {                                            }
                    { {CONTROL FOOTING}   {data-name-3}          }
                    { {CF             }   {FINAL      }          }
                    {                                            }
                    { {PAGE FOOTING   }                          }
                    { {PF             }                          }
                    {                                            }
                    { {REPORT FOOTING }                          }
                    { {RF             }                          }

[; [USAGE IS]         {DISPLAY  }] .
                      {DISPLAY-1}
```

Option 3:

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

[; LINE NUMBER IS  {integer-1 [ON NEXT PAGE]}]
                   {PLUS integer-2          }

[; [USAGE IS] {DISPLAY   }]
              {DISPLAY-1 }
```

Option 4:

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

           [; BLANK WHEN ZERO]

           [; COLUMN NUMBER IS  integer-3]

           [; GROUP INDICATE]

           [;{JUSTIFIED} RIGHT]
             {JUST     }

           [; LINE NUMBER IS {integer-1  ON NEXT PAGE}]
                             {PLUS integer-2         }

            ;{PICTURE} IS character-string
             {PIC    }

           {  ;SOURCE IS {TODAYS-DATE }                                  }
             {           {identifier-1}                                  }
             {;SUM  identifier-2 [, identifier-3] ...                    }
             {      [UPON  data-name-2 [, data-name-3] ...]}...          }
             {      [RESET ON {FINAL      }]                             }
             {                {data-name-4}                              }
             {                                                           }
             {;{VA   } IS  literal-1                                     }
             {  {VALUE}                                                  }

           [; [USAGE IS] {DISPLAY   }]
                         {DISPLAY-1 }
```

The Report Group Description entry can appear only in the Report Section.

Except for the data-name clause, which when present must immediately follow the level-number, the clauses may be written in any sequence.

In all formats above, integers must be greater than zero.

Level-number in option 3 may be any integer from 02 to 48 inclusive. Level-number in option 4 may be any integer from 02 to 49 inclusive.

The description of a report group may consist of one, two or three hierarchic levels. The first entry of a report group must be an option 2 entry. An option 3 entry describes a single line of the report group and must be followed immediately by option 4 entries describing the printable items for the line. Option 4 entries, in addition to describing a single printable item of a line, may also be used to describe a line which contains only one printable item.

An entry that contains a LINE NUMBER clause must not have a subordinate entry that also contains a LINE NUMBER clause.

Data-name-1 of an option 2 entry may be referenced only by a GENERATE statement, the UPON phrase of a SUM clause, a USE BEFORE REPORTING sentence or as a sum counter qualifier. Data-name-1 is required in an option 2 entry only when:

a. A DETAIL report group is referenced by a GENERATE statement,
b. A DETAIL report group is referenced by the UPON phrase of a SUM clause,
c. A report group is referenced in a USE BEFORE REPORTING sentence,
d. The name of a CONTROL FOOTING report group is used to qualify a reference to a sum counter.

In an option 3 entry, data-name-1 is optional. If present, it may be used only to qualify a sum counter reference. An option 3 entry must contain at least one of the optional clauses.

In an option 4 entry:

a. A GROUP INDICATE clause may appear only in a TYPE DETAIL report group.
b. A SUM clause may appear only in a TYPE CONTROL FOOTING report group.
c. An entry that contains a COLUMN NUMBER clause but no LINE NUMBER clause must be subordinate to an entry that contains a LINE NUMBER clause.
d. Data-name-1 is optional and may be referenced only if the entry defines a SUM counter.
e. A LINE-NUMBER clause must not be the only clause specified.
f. An entry that contains a VALUE clause must also have a COLUMN NUMBER clause.

Figure 11-3 shows all permissible clause combinations for an option 4 entry. The table is read from left to right along the selected row.

An "M" indicates that the presence of the clause is mandatory.

A "P" indicates that the presence of the clause is permitted, but not required.

A blank indicates that the clause is not permitted.

PIC	COLUMN	SOURCE	SUM	VALUE	JUST	BLANK WHEN ZERO	GROUP INDICATE	USAGE	LINE
M			M						P
M	M		M			P		P	P
M	P	M			P		P	P	P
M	P	M				P	P	P	P
M	M			M	P		P	P	P

Figure 11-3. Permissible Clause Combinations in Option 3 Report Group Description Entries

The JUSTIFIED, PICTURE, and BLANK WHEN ZERO clauses are described in Section 6.

COLUMN NUMBER

The COLUMN NUMBER clause identifies a printable item and specifies the column number position of the item on the print line.

COLUMN NUMBER IS integer-6

The COLUMN NUMBER clause can only be specified at the elementary level. When this clause is used, it should appear in or be subordinate to an entry that contains a LINE NUMBER clause.

Integer-6 must be greater than zero. Integer-6 specifies the leftmost character position of the printable item. Within a given print line, printable items must be defined in ascending column number order such that each character defined occupies a unique position.

The COLUMN NUMBER clause indicates that the object of a SOURCE clause or the object of a VALUE clause or the sum counter defined by a SUM clause is to be printed with its leftmost character position indicated by integer-6.

The first or leftmost character of a print line is column number 1.

The absence of a COLUMN NUMBER clause indicates that the entry is not to be printed.

Space characters are automatically provided for all positions of a print line which are not occupied by printable items.

GROUP INDICATE

The GROUP INDICATE clause indicates that this elementary item is to be produced only on the first occurrence of the item after any control or page break.

GROUP INDICATE

The GROUP INDICATE clause may only appear in a DETAIL report group at the elementary item level within an entry that defines a printable item.

If a GROUP INDICATE clause is specified, it causes the SOURCE or VALUE clauses to be ignored and spaces provided except:

a. On the first printing of the DETAIL report group in the report or
b. On the first printing of the DETAIL report group after a page advance, or
c. On the first printing of the DETAIL report group after every control break.

If the report description entry specifies neither a PAGE clause nor a CONTROL clause, then a GROUP INDICATE printable item is printed the first time its DETAIL is printed after the INITIATE statement is executed. Thereafter, spaces are supplied for indicated items with SOURCE or VALUE clauses.

LINE NUMBER

The LINE NUMBER clause specifies vertical positioning information for its report group.

```
LINE NUMBER IS  {integer-1          [ON NEXT PAGE]}
                {PLUS integer-2                   }
```

Integer-1 and integer-2 must not exceed three significant digits in length.

Neither integer-1 nor integer-2 may be specified in such a way as to cause any line of a report group to be presented outside of the vertical subdivision of the page designated for that report group type, as defined by the PAGE clause.

Within a given Report Group Description entry, an entry that contains a LINE NUMBER clause must not contain a subordinate entry that also contains a LINE NUMBER clause.

Within a given Report Group Description entry, all absolute LINE NUMBER clauses must precede all relative LINE NUMBER clauses.

Within a given Report Group Description entry, successive absolute LINE NUMBER clauses must specify integers that are in ascending order. The integers need not be consecutive.

If the PAGE clause is omitted from a given Report Group Description entry, only relative LINE NUMBER clauses can be specified in any Report Group Description entry within that report.

Within a given Report Group Description entry a NEXT PAGE phrase can appear only once and, if present, must be in the first LINE NUMBER clause in that Report Group Description entry. A LINE NUMBER clause with the NEXT PAGE phrase can appear only in the description of body groups and in a REPORT FOOTING report group.

Every entry that defines a printable item must either contain a LINE NUMBER clause, or be subordinate to an entry that contains a LINE NUMBER clause.

The first LINE NUMBER clause specified within a PAGE FOOTING report group must be an absolute LINE NUMBER clause.

A LINE NUMBER clause must be specified to establish each print line of a report group.

The vertical positioning specified by a LINE NUMBER clause occurs before printing the line established by that LINE NUMBER clause.

Integer-1 specifies an absolute line number. An absolute line number specifies the line number on which the print line is printed.

Integer-2 specifies a relative line number. If a relative LINE NUMBER clause is specified then the line number on which its print line is printed is determined by calculating the sum of the line number on which the previous print line of the report group was printed and integer-2 of the relative LINE NUMBER clause.

The NEXT PAGE phrase specifies that the report group is to be presented beginning on the indicated line number on a new page.

NEXT GROUP

The NEXT GROUP clause specifies information for vertical positioning of a page following the presentation of the last line of a report group.

```
NEXT GROUP IS  { integer-1          }
               { PLUS integer-2     }
               { NEXT PAGE          }
```

A report group entry must not contain a NEXT GROUP clause unless the description of that report group contains at least one LINE NUMBER clause.

Integer-1 and integer-2 must not exceed the value 255.

If the PAGE clause is omitted from the Report Description entry only a relative NEXT GROUP clause may be specified in any Report Group Description entry within that report.

The NEXT PAGE phrase of the NEXT GROUP clause must not be specified in a PAGE FOOTING report group.

The NEXT GROUP clause must not be specified in a REPORT FOOTING report group or in a PAGE HEADING report group.

Any positioning of the page specified by the NEXT GROUP clause takes place after the printing of the report group in which the clause appears.

The vertical positioning information supplied by the NEXT GROUP clause is interpreted along with information from the TYPE and PAGE clauses, and the value in LINE-COUNTER, to determine a new value for LINE-COUNTER.

The NEXT GROUP clause is ignored when it is specified on a CONTROL FOOTING report group that is at a level other than the highest level at which a control break is detected.

The NEXT GROUP clause of a body group refers to the next body group to be printed, and therefore can affect the location at which the next body group is printed. The NEXT GROUP clause of a REPORT HEADING report group can affect the location at which the PAGE HEADING report group is printed. The NEXT GROUP clause of a PAGE FOOTING report group can affect the location at which the REPORT FOOTING report group is printed.

SOURCE

The SOURCE clause identifies the sending data item that is moved to an associated printable item defined within a Report Group Description entry.

$$\underline{\text{SOURCE}} \text{ IS } \left\{ \begin{matrix} \underline{\text{TODAYS-DATE}} \\ \text{identifier-1} \end{matrix} \right\}$$

Identifier-1 may be defined in any section of the Data Division. If identifier-1 is a Report Section item it can only be:

a. PAGE-COUNTER, or
b. LINE-COUNTER, or
c. A sum counter of the report within which the SOURCE clause appears.

Identifier-1 specifies the sending data item of the implicit MOVE statement executed to move identifier-1 to the printable item. Identifier-1 must be defined such that it conforms to the rules for sending items in the MOVE statement. Identifier-1 may be any special register, attribute, intrinsic function or identifier.

The print lines of a report group are formatted just prior to presenting the report group. It is at this time that the implicit MOVE statements specified by SOURCE clauses are executed.

SUM

The SUM clause establishes a sum counter and names the data items to be summed.

```
{SUM identifier-1    [, identifier-2]    . . .

    [UPON data-name-1    [, data-name-2]    . . . ]}    . . .

    [RESET ON {data-name-3}]
              {FINAL      }
```

Identifier-1 and identifier-2 must be defined as numeric data items. When defined in the Report Section, identifier-1 and identifier-2 must be the names of sum counters. If the UPON phrase is omitted, any identifiers in the associated SUM clause which are themselves sum counters must be defined either in the same report group that contains this SUM clause or in a report group which is at a lower level in the control hierarchy of this report. If the UPON phrase is specified, any identifiers in the associated SUM clause must not be sum counters.

Data-name-1 and data-name-2 must be the names of DETAIL report groups described in the same report as the CONTROL FOOTING report group in which the SUM clause appears. Data-name-1 and data-name-2 may be qualified by a report-name.

A SUM clause can appear only in the description of a CONTROL FOOTING report group.

Data-name-3 must be one of the data-names specified in the CONTROL clause for this report. Data-name-3 must not be a lower level control than the associated control for the report group in which the RESET phrase appears.

FINAL, if specified in the RESET phrase, must also appear in the CONTROL clause for this report.

The highest permissible qualifier of a sum counter is the report-name.

The SUM clause establishes a sum counter. The sum counter is a numeric data item with an operational sign. At object time each of the values identifier-1, identifier-2 . . . is added directly into the sum counter. This addition is performed under the rules of the ADD statement.

The size of the sum counter is equal to the number of receiving character positions specified by the PICTURE clause that accompanies the SUM clause in the description of the elementary item.

Only one sum counter exists for an elementary report entry regardless of the number of SUM clauses specified in the elementary report entry.

If the elementary report entry for a printable item contains a SUM clause, the sum counter serves as a source data item. The data contained in the sum counter, is moved according to the rules of the MOVE statement, to the printable item for printing.

If the data-name appears as the subject of an elementary report entry that contains a SUM clause, the data-name is the name of the sum counter; the data-name is not the name of the printable item that the entry may also define.

It is permissible for Procedure Division statements to alter the contents of sum counters.

Addition of the identifiers into sum counters is performed during the execution of GENERATE and TERMINATE statements. There are three categories of sum counter incrementing called subtotalling, crossfooting, and rolling forward. Subtotalling is accomplished during execution of GENERATE statements only, after any control break processing but before processing of the DETAIL report group. Crossfooting and rolling forward are accomplished during the processing of CONTROL FOOTING report groups.

The UPON phrase provides the capability to accomplish selective subtotalling for the DETAIL report groups named in the phrase.

Each individual addend is added into the sum counter at a time that depends upon the characteristics of the addend.

a. When the addend is a sum counter defined in the same CONTROL FOOTING report group, then the accumulation of that addend into the sum counter is termed crossfooting. Crossfooting occurs when a control break takes place and at the time the CONTROL FOOTING report group is processed. Crossfooting is performed according to the sequence in which sum counters are defined within the CONTROL FOOTING report group. That is, all crossfooting into the first sum counter defined in the CONTROL FOOTING report group is completed, and then all cross-footing into the second sum counter defined in the CONTROL FOOTING report group is completed. This procedure is repeated until all cross-footing operations are completed.

b. When the addend is a sum counter defined in a lower level CONTROL FOOTING report group, then the accumulation of that addend into the sum counter is termed rolling forward. A sum counter in a lower level CONTROL FOOTING report group is rolled forward when a control break occurs and at the time that the lower level CONTROL FOOTING report group is processed.

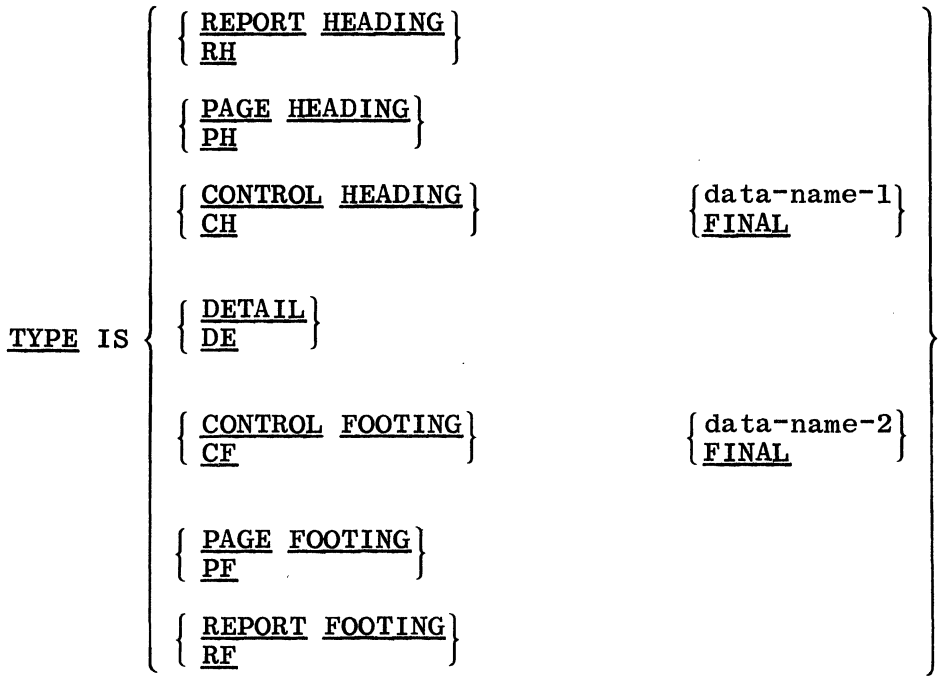
c. When the addend is not a sum counter the accumulation into a sum counter of such an addend is called subtotalling. If the SUM clause contains the UPON phrase, the addends are subtotalled when a GENERATE statement for the designated DETAIL report group is executed. If the SUM clause does not contain the UPON phrase, the addends which are not sum counters are subtotalled when any GENERATE data-name statement is executed for the report in which the SUM clause appears.

If two or more of the identifiers specify the same addend, then the addend is added into the sum counter as many times as the addend is referenced in the SUM clause. It is permissible for two or more of the data-names to specify the same DETAIL report group. When a GENERATE data-name statement for such a DETAIL report group is given, the incrementing occurs repeatedly, as many times as data-name appears in the UPON phrase.

In the absence of an explicit RESET phrase, a sum counter will be set to zero at the time of processing the CONTROL FOOTING report group within which the sum counter is defined. If an explicit RESET phrase is specified, then the sum counter is set to zero at the time of processing the designated level of the control hierarchy. Sum counters are initially set to zero during the execution of the INITIATE statement for the report containing the sum counter.

TYPE

The TYPE clause specifies the particular type of report group that is described by this entry and indicates the time at which the report group is to be processed.



REPORT HEADING, PAGE HEADING, CONTROL HEADING FINAL, CONTROL FOOTING FINAL, PAGE FOOTING, and REPORT FOOTING report groups may each appear no more than once in the description of a report.

PAGE HEADING and PAGE FOOTING report groups may be specified only if a PAGE clause is specified in the corresponding Report Description entry.

Data-name-1, data-name-2 and FINAL, if present, must be specified in the CONTROL clause of the corresponding Report Description entry. At most, one CONTROL HEADING report group and one CONTROL FOOTING report group can be specified for each data-name or FINAL in the CONTROL clause of the Report Description entry. However, neither a CONTROL HEADING report group nor a CONTROL FOOTING report group is required for a data-name or FINAL specified in the CONTROL clause of the Report Description entry.

In CONTROL FOOTING, PAGE HEADING, PAGE FOOTING, and REPORT FOOTING report groups, SOURCE clauses and USE statements must not reference any of the following:

a. Group data items containing a control data item.
b. Data items subordinate to a control data item.
c. A redefinition or renaming of any part of a control data item.

In PAGE HEADING and PAGE FOOTING report groups, SOURCE clauses and USE statements must not reference control data-names.

When a GENERATE report-name statement is specified in the Procedure Division, the corresponding Report Description entry must include no more than one DETAIL report group. If no GENERATE data-name statements are specified for such a report, a DETAIL report group is not required.

The description of a report must include at least one body group.

DETAIL report groups are processed as a direct result of GENERATE statements. If a report group is other than TYPE DETAIL, its processing is an automatic function.

The REPORT HEADING phrase specifies a report group that is processed only once per report, as the first report group of that report. The REPORT HEADING report group is processed during the execution of the chronologically first GENERATE statement for that report.

The PAGE HEADING phrase specifies a report group that is processed as the first report group on each page of that report except under the following conditions:

a. A PAGE HEADING report group is not processed on a page that is to contain only a REPORT HEADING report group or only a REPORT FOOTING report group.
b. A PAGE HEADING report group is processed as the second report group on a page when it is preceded by a REPORT HEADING report group that is not to be printed on a page by itself.

The CONTROL HEADING phrase specifies a report group that is processed at the beginning of a control group for a designated control data-name or, in the case of FINAL, is processed during the execution of the chronologically first GENERATE statement for that report. During the execution of any GENERATE statement at which a control break is detected, any CONTROL HEADING report groups associated with the highest control level of the break and lower levels are processed.

The DETAIL phrase specifies a report group that is processed when a corresponding GENERATE statement is executed.

The CONTROL FOOTING phrase specifies a report group that is processed at the end of a control group for a designated control data-name. In the case of FINAL, the CONTROL FOOTING report group is processed only once per report as the last body group of that report. During the execution of any GENERATE statement in which a control break is detected, any CONTROL FOOTING report group associated with the highest level of the control break or more minor levels is printed. All CONTROL FOOTING report groups are printed during the execution of the TERMINATE statement if there has been at least one GENERATE statement executed for the report.

The PAGE FOOTING phrase specifies a report group that is processed as the last report group on each page except under the following conditions:

a. A PAGE FOOTING report group is not processed on a page that is to contain only a REPORT HEADING report group or only a REPORT FOOTING report group.
b. A PAGE FOOTING report group is processed as the second to last report group on a page when it is followed by a REPORT FOOTING report group that is not to be processed on a page by itself.

The REPORT FOOTING phrase specifies a report group that is processed only once per report as the last report group of that report. The REPORT FOOTING report group is processed during the execution of a corresponding TERMINATE statement, if there has been at least one GENERATE statement executed for the report.

The sequence of steps executed when a REPORT HEADING, PAGE HEADING, CONTROL HEADING, PAGE FOOTING, or REPORT FOOTING report group is processed is described below.

a. If there is a USE BEFORE REPORTING procedure that references the data-name of the report group, the USE procedure is executed.
b. If the report group is not printable, there is no further processing to be done for the report group.
c. Otherwise, the print lines are formatted and printed according to the rules for that type of report group.

The sequence of steps executed when a CONTROL FOOTING report group is processed is described below.

The GENERATE rules specify that when a control break occurs, the CONTROL FOOTING report groups beginning at the minor level, and proceeding upwards are processed through the level at which the highest control break was sensed. In this regard, it should be noted that even though no CONTROL FOOTING report group has been defined for a given control data-name, the step described in paragraph (f) below will still be executed if a RESET phrase within the report description specifies that control data-name.

a. Sum counters are crossfooted, i.e., all sum counters defined in this report group that are operands of SUM clauses in the same report group are added to their sum counters.
b. Sum counters are rolled forward, i.e., all sum counters defined in the report group that are operands of SUM clauses in higher level CONTROL FOOTING report groups are added to the higher level sum counters.
c. If there is a USE BEFORE REPORTING procedure that references the data-name of the report group, the USE procedure is executed.
d. If the report group is not printable, the step described in paragraph (f) is executed next.
e. Otherwise, the print lines are formatted and the report group is printed according to the rules for CONTROL FOOTING report groups.
f. Then those sum counters that are to be reset when processing this level in the control hierarchy are reset.

The DETAIL report group processing that is executed in response to a GENERATE data-name statement is described in paragraphs (a) thru (e) below.

When the description of a report includes exactly one DETAIL report group, the detail-related processing that is executed in response to a GENERATE report-name statement is described in paragraph (a) thru paragraph (d) below. These steps are performed as though a GENERATE data-name statement were being executed.

When the description of a report includes no DETAIL report groups, the detail-related processing that is executed in response to a GENERATE report-name statement is described in paragraph (a) below. This step is performed as though the description of the report included exactly one DETAIL report group, and a GENERATE data-name statement were being executed.

a. Any subtotalling is performed that has been designated for the DETAIL report group.
b. If there is a USE BEFORE REPORTING procedure that refers to the data-name of the report group, the USE procedure is executed.
c. If the report group is not printable there is no further processing done for the report group.
d. If the DETAIL report group is being processed as a consequence of a GENERATE report-name statement, there is no further processing done for the report group.
e. Otherwise, the print lines are formatted and the report group printed according to the rules for DETAIL report groups.

When a CONTROL HEADING, CONTROL FOOTING, or DETAIL report is being processed, as previously described, processing of that body group may have to be interrupted after determining that the body group is to be printed, and execute a page advance (and process PAGE FOOTING and PAGE HEADING report groups) before actually printing the body group.

During control break processing, the values of control data items used to detect a given control break are referred to as prior values.

a. During control break processing of a CONTROL FOOTING report group, any references to control data items in a USE procedure or SOURCE clause associated with that CONTROL FOOTING report group are supplied with prior values.
b. When a TERMINATE statement is executed, the prior control data item values are made available to SOURCE clause or USE procedure references in CONTROL FOOTING and REPORT FOOTING report groups as though a control break had been detected in the highest control data-name.

c. All other data item references within report groups and their USE procedures access the current values that are contained within the data items at the time the report group is processed.

USAGE

[USAGE IS] {DISPLAY / DISPLAY-1}

The USAGE clause may be used at either the 01 or elementary level; however, the USAGE of all report groups and their elementary items must be the same as the USAGE of the file on which the report will be written. (See Section 6 for additional information.)

VALUE

The VALUE clause defines the value of Report Section printable items.

VALUE IS literal

See Section 6 for a discussion of the VALUE clause. Only option 1 of the VALUE clause is permitted in the Report Section.

PROCEDURE DIVISION

INITIATE

The INITIATE statement begins processing of a report. The format is as follows:

```
INITIATE   report-name-1   [, report-name-2] ...
```

Each report-name must be defined by a report description entry in the REPORT SECTION of the DATA DIVISION.

The INITIATE statement resets all data-name entries that contain SUM clauses associated with this report.

The PAGE-COUNTER register, if specified, is set to one (1) during the execution of the INITIATE STATEMENT. If a different starting value for the associated PAGE-COUNTER other than (1) is desired, the programmer may reset the counter after the completion of the execution of the INITIATE statement.

The LINE-COUNTER register, if specified, is set to zero prior to or during the execution of the INITIATE statement.

The INITIATE statement does not open the file with which the report is associated; however, the associated file must be open at the time the INITIATE statement is executed.

A second INITIATE for a particular report-name may not be executed unless a TERMINATE statement has been executed for that report-name subsequent to the first INITIATE statement.

GENERATE

The GENERATE statement links the PROCEDURE DIVISION to the REPORT WRITER (described in the REPORT SECTION of the DATA DIVISION) at process time. The statement format is as follows:

GENERATE identifier

Identifier represents a TYPE DETAIL report group or an RD entry.

If identifier represents the name of a TYPE DETAIL report group, the GENERATE statement does all the automatic operations within the REPORT WRITER and produces an actual output DETAIL report group, at process time, on the output medium. This is called detail reporting.

If identifier represents the name of an RD entry, the GENERATE statement does all the automatic operations of the REPORT WRITER and updates the FOOTING report group(s) within a particular report description without producing an actual DETAIL report group associated with the report. In this case, all SUM counters associated with the report description are algebraically incremented. This is called summary reporting. For summary reporting, there may be no more than 1 TYPE DETAIL report group, there must be at least 1 body group, and the CONTROL clause must be specified for the report.

A GENERATE statement implicitly produces in both detail and summary reporting the following automatic operations (if defined):

a. Steps and tests LINE-COUNTER and/or PAGE-COUNTER to produce appropriate PAGE FOOTING and/or PAGE HEADING report groups.
b. Recognizes any specified control breaks to produce appropriate CONTROL FOOTING and/or CONTROL HEADING reporting groups.
c. Accumulates into the SUM counters all specified identifier(s). Resets the SUM counters on an associated control break. Performs an updating procedure between control break levels for each set of SUM counters.
d. Executes any specified routines defined by a USE statement before generation of the associated report groups(s).

During the execution of the first GENERATE statement, the following report groups associated with the report, if specified, are produced in the following order:

a. REPORT HEADING report group.
b. PAGE HEADING report group.

c. All CONTROL HEADING report groups in the following order: final, major, minor.
d. The DETAIL report group, if specified in the GENERATE statement.

If a control break is recognized at the time of execution of a GENERATE statement (other than the first executed for a report), all CONTROL FOOTING report groups specified for the report are produced from the minor report group, up to and including the report group specified for the identifier which caused the control break. Next, the CONTROL HEADING report group(s) specified for the report, from the report group specified for the identifier that caused the control breakdown to the minor report group, are produced in that order. The DETAIL report group specified in the GENERATE statement is then produced.

Data is moved to the data item in the report group description entry of the REPORT section. This data is edited under control of the REPORT WRITER, according to the same rules for movement and editing as described for the MOVE statement.

GENERATE statements for a report can be executed only after an INITIATE statement for the report has been executed and before a TERMINATE statement for the report has been executed.

TERMINATE

The TERMINATE statement terminates the processing of a report. The statement format is as follows:

TERMINATE report-name-1 [, report-name-2] ... .

Each report-name given in a TERMINATE statement must be defined by an RD entry in the REPORT SECTION of the DATA DIVISION.

The TERMINATE statement produces all the CONTROL FOOTING groups associated with this report as if a control break had just occurred at the highest level, and completes the REPORT WRITER functions for the named reports. The TERMINATE statement also produces the last PAGE FOOTING and the REPORT FOOTING report groups associated with this report.

If no GENERATE statements have been executed for a report during the interval between the execution of an INITIATE statement and a TERMINATE statement for the same report, associated FOOTING groups will not be produced.

Appropriate PAGE HEADING and/or PAGE FOOTING report groups are prepared in their respective order for the report description.

A second TERMINATE for a particular report may not be executed unless a second INITIATE statement has been executed for a report-name. If a TERMINATE statement has been executed for a report, a GENERATE statement for that report must not be executed unless an intervening INITIATE statement for that report is executed.

The TERMINATE statement does not close the file with which the report is associated; a CLOSE statement for the file must be given by the user. However, the associated file must be open at the time the TERMINATE statement is executed. The TERMINATE statement performs REPORT WRITER functions for individually described reports analogous to the input/output functions that the CLOSE statement performs for individually described files.

SOURCE clauses used in the CONTROL FOOTING FINAL or REPORT FOOTING report groups refer to the values of the items at the execution time of the TERMINATE statement.

USE

A special format of the USE statement may be used to specify PROCEDURE DIVISION statements to be executed just before a report group named in the REPORT SECTION of the DATA DIVISION is produced. The statement format is as follows:

USE BEFORE REPORTING identifier-1

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.

Identifier-1 represents a report group named in the REPORT SECTION of the DATA DIVISION. An identifier must not appear in more than one USE statement.

No REPORT WRITER statement (GENERATE, INITIATE, OR TERMINATE) may be written in a procedural paragraph or paragraphs following the USE sentence in the DECLARATIVE portion.

The USE statement itself is never executed; rather, it defines the conditions calling for the execution of the USE procedures.

The designated procedures are executed by the REPORT WRITER just before the named report group is produced, regardless of control break associations with report groups.

A USE BEFORE REPORTING must not alter the value of any control data item.

SAMPLE REPORT WRITER PROGRAM

The following pages present a sample of a program which uses the report writer feature.

Figure 11-4 is a sample report. The numbers on the far right hand side indicate the report group type which caused the line to be printed as follows:

1. Report heading
2. Page heading
3. Detail
4. Control footing GRADE
5. Control footing DAYY
6. Control footing MONTH
7. Page footing
8. Report footing

Figure 11-5 shows the program used to prepare the report and also has numbers at the far right hand side in the same manner as on the sample report.

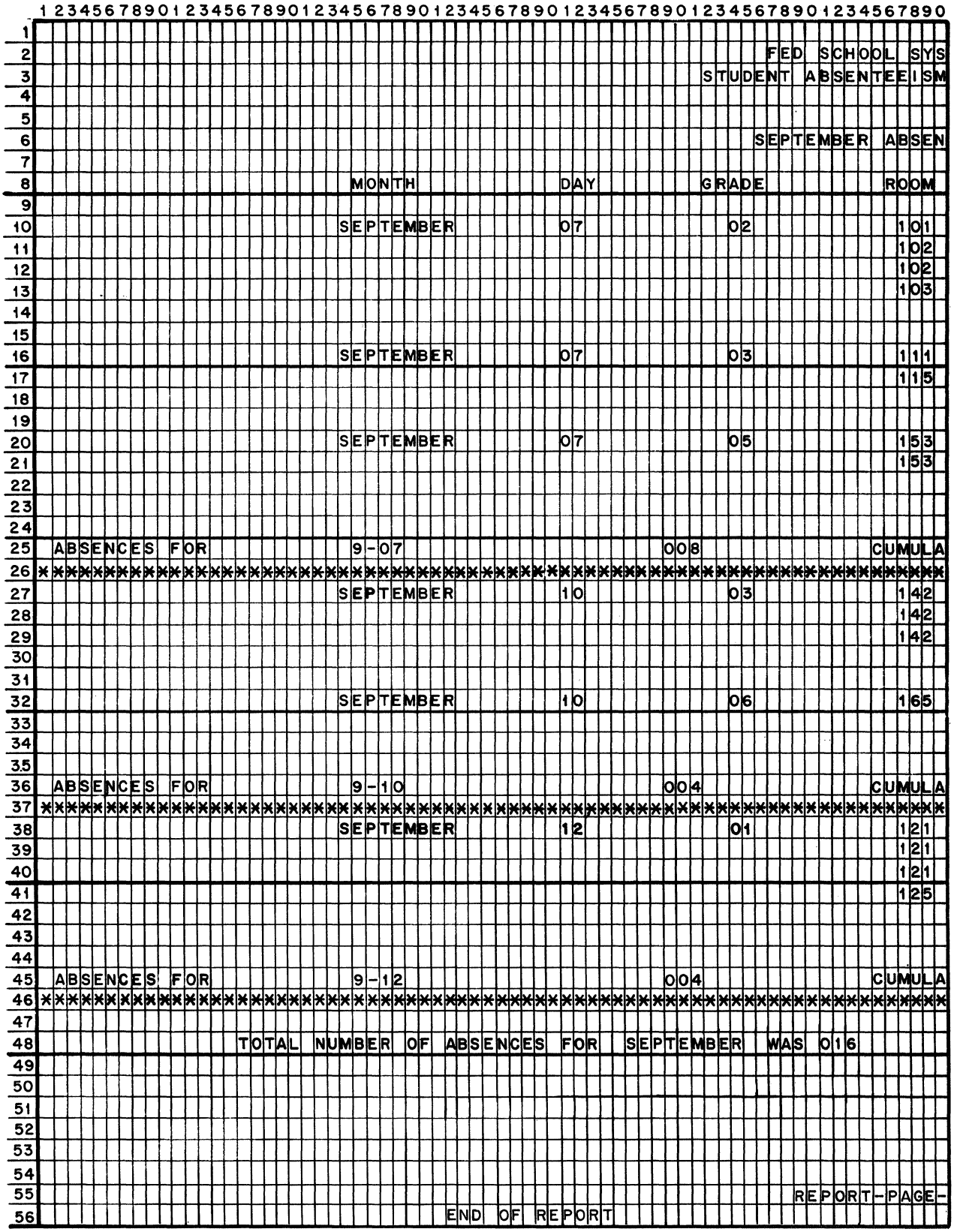
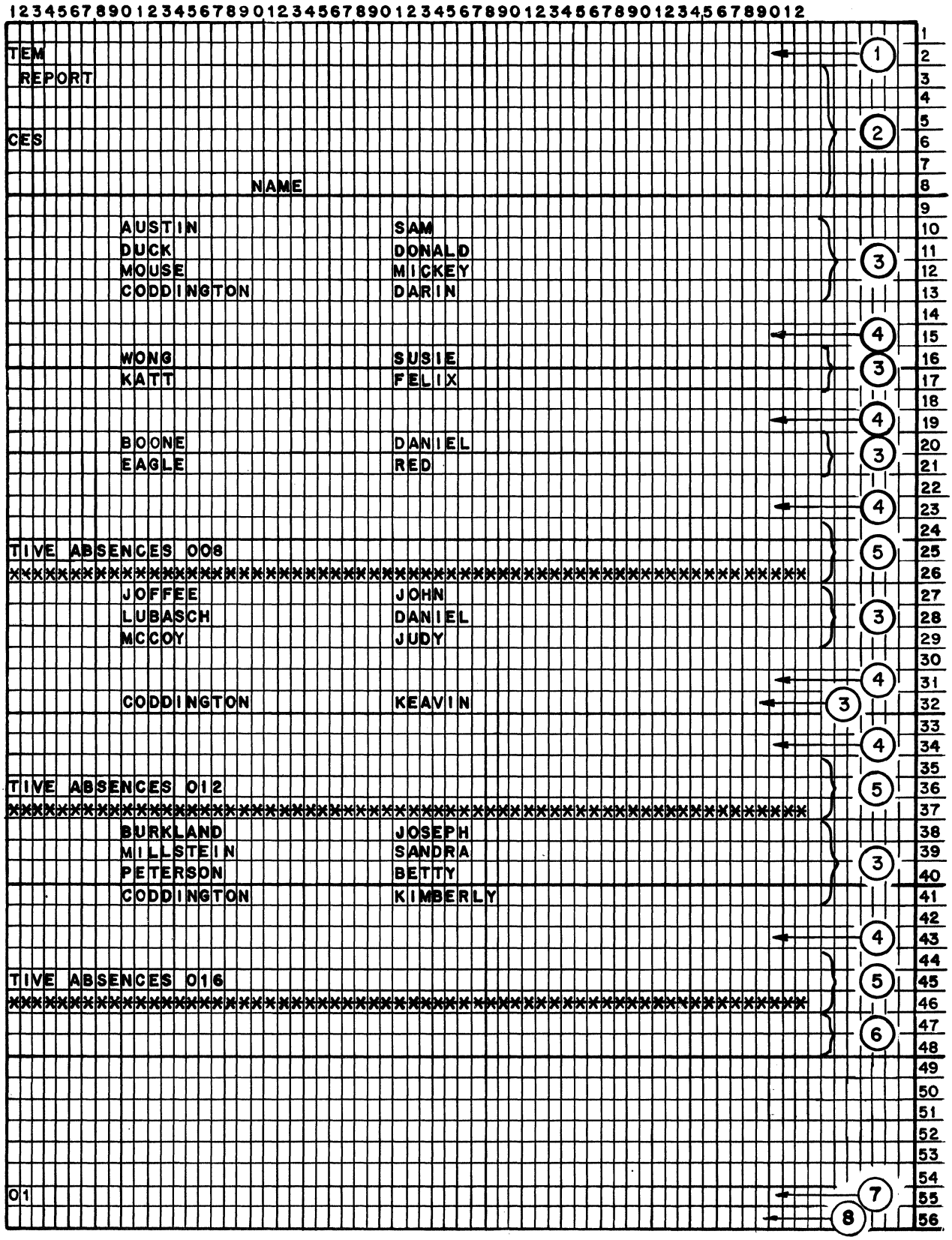


Figure 11-4. Sample Report Writer Report


1234567890123456789012345678901234567890123456789012345678901234567890123456789012
TEM
REPORT
CES
NAME
AUSTIN
SAM
DUCK
DONALD
MOUSE
MICKEY
CODDINGTON
DARIN
WONG
SUSIE
KATT
FELIX
BOONE
DANIEL
EAGLE
RED
TIVE ABSENCES 008
JOFFEE
JOHN
LUBASCH
DANIEL
MCCOY
JUDY
CODDINGTON
KEAVIN
TIVE ABSENCES 012
BURKLAND
JOSEPH
MILLSTEIN
SANDRA
PETERSON
BETTY
CODDINGTON
KIMBERLY
TIVE ABSENCES 016
01
1
2
3
4
3
4
3
4
5
3
4
3
4
5
3
4
5
6
7
8

```
001000 IDENTIFICATION DIVISION.
001100 PROGRAM-ID. FED-SCHOOL-SYSTEM, FRB-NY; BERKOWITZ.
001200 ENVIRONMENT DIVISION.
001300 INPUT-OUTPUT SECTION.
001400 FILE-CONTROL.
001500     SELECT PENNI ASSIGN TO SORT DISK.
001600     SELECT INFILE ASSIGN TO CARD-READER.
001700     SELECT REPORTFILE ASSIGN TO PRINTER.
001800 DATA DIVISION.
001900 FILE SECTION.
002000 FD  INFILE            01  IN-REC  SZ 80.
002100 SD  PENNI.            01  FROMM.
002200     02  FILLER      PICTURE XX.
002300     02  STUDENT.
002400        03  NAME-L   PICTURE X(30).
002500        03  NAME-F   PICTURE X(10).
002600     02  FILLER      PICTURE XX.
002700     02  GRADE       PICTURE 99.
002800     02  FILLER      PICTURE XX.
002900     02  ROOM        PICTURE 999.
003000     02  FILLER      PICTURE 99.
003100     02  MONTH       PICTURE 99.
003200     02  DAYY        PICTURE 99.
003300     02  YR          PICTURE 99.
003400     02  FILLER      PICTURE X(20).
003500     02  TAL         PICTURE 9.
003600 FD  REPORTFILE   REPORT IS ABS-REPORT.
003700 WORKING-STORAGE SECTION.
003800 77  SAVED-MONTH     PICTURE 99 VALUE IS 0.
003900 77  CONTINUED       PICTURE X(11) VALUE IS SPACE.
004000 77  ABSS PIC X(8)  VALUE "ABSENCES".
004100 77  CA PIC X(19)   VALUE "CUMULATIVE ABSENCES".
004200 77  TAL-CTR  COMP-1  PIC 9999.
004300 01  HEAD-1.
004400     02  FILLER  PIC X(24) VALUE SPACES.
004500     02 HEAD-LINE  PIC X(72) VALUE "MONTH                DAY
004600-       " GRADE         ROOM                      NAME".
004700     02  FILLER  PIC X(36) VALUE SPACES.
004800 01  MONTH-TABLE.
004900     02  MONTH-1.
005000         03 FILLER PICTURE A(9) VALUE IS "JANUARY  ".
005100         03 FILLER PICTURE A(9) VALUE IS "FEBRUARY ".
005200         03 FILLER PICTURE A(9) VALUE IS "MARCH    ".
005300         03 FILLER PICTURE A(9) VALUE IS "APRIL    ".
005400         03 FILLER PICTURE A(9) VALUE IS "MAY      ".
005500         03 FILLER PICTURE A(9) VALUE IS "JUNE     ".
005600         03 FILLER PICTURE A(9) VALUE IS "JULY     ".
005700         03 FILLER PICTURE A(9) VALUE IS "AUGUST   ".
005800         03 FILLER PICTURE A(9) VALUE IS "SEPTEMBER".
005900         03 FILLER PICTURE A(9) VALUE IS "OCTOBER ".
006000         03 FILLER PICTURE A(9) VALUE IS "NOVEMBER".
006100         03 FILLER PICTURE A(9) VALUE IS "DECEMBER".
006200         03 FILLER PICTURE A(9) VALUE SPACES.
006300     02  MONTH-2 REDEFINES MONTH-1.
006400           03  MONTHNAME  PICTURE A(9) OCCURS 13 TIMES.
```

Figure 11-5. Sample Report Writer Program

```
006500 REPORT SECTION.
006600 RD  ABS-REPORT   CONTROLS ARE FINAL, MONTH, DAYY, GRADE
006700     PAGE LIMIT IS 56 LINES      HEADING 2
006800     FIRST DETAIL 10   LAST DETAIL 45      FOOTING 55.
006900 01  TYPE IS REPORT HEADING.
007000     02  LINE NUMBER IS 2   COLUMN 57      PIC X(17)
007100        VALUE "FED SCHOOL SYSTEM".
007200 01  PAGE-HEAD    TYPE IS PAGE HEADING.
007300     02  LINE NUMBER IS 3   COLUMN 52      PIC X(26)
007400        VALUE "STUDENT  ABSENTEISM REPORT".
007500     02  LINE NUMBER IS 6.
007600        03  COLUMN IS 56    PIC X(9)
007700            SOURCE IS MONTHNAME OF MONTH-2(MONTH).
007800        03  COLUMN IS 66    PIC X(8)       SOURCE IS ABSS.
007900        03  COLUMN IS 76    PIC X(11)      SOURCE IS CONTINUED.
008000     02  LINE IS 8.
008100        03  COLUMN IS 1     PIC X(132)     SOURCE HEAD-1.
008200 01  DETAIL-LINE TYPE IS DETAIL LINE NUMBER IS PLUS 1.
008300     02  COLUMN IS 24 GROUP INDICATE PIC X(9)
008400         SOURCE IS MONTHNAME OF MONTH-2(MONTH).
008500     02  COLUMN IS 41 GROUP INDICATE PICTURE IS 99
008600         SOURCE IS DAYY.
008700     02  COLUMN IS 54 GROUP INDICATE PIC  99 SOURCE IS GRADE.
008800     02  COLUMN IS 67 PIC 999 SOURCE IS ROOM.
008900     02  COLUMN IS 80 PIC X(20) SOURCE IS  NAME-L.
009000     02  COLUMN IS 101 PIC X(10) SOURCE IS NAME-F.
009100 01  TYPE IS CONTROL FOOTING GRADE.
009200     02  LINE NUMBER IS PLUS 2.
009300        03  COLUMN 1 PIC X(132) VALUE SPACE.
009400 01  TESTER   TYPE IS CONTROL FOOTING DAYY.
009500     02  LINE NUMBER IS PLUS 2.
009600        03  COLUMN 2 PIC X(12)             VA "ABSENCES FOR".
009700        03  COLUMN 24  PICTURE Z9 SOURCE SAVED-MONTH.
009800        03  COLUMN 26  PICTURE X VALUE "-".
009900        03  COLUMN 27  PICTURE 99 SOURCE DAYY.
010000        03  NO-ABS          COLUMN 49      PIC 999   SUM TAL.
010100        03  COLUMN 65 PIC X(19) SOURCE CA.
010200        03  COLUMN 85  PIC 999 SUM TAL RESET ON FINAL.
010300     02  LINE PLUS 1 COLUMN 1    PIC X(132)     VA ALL "*".
010400 01  TYPE CONTROL FOOTING MONTH
010500     LINE PLUS 2 NEXT GROUP NEXT PAGE.
010600     02 COLUMN 16 PIC X(28) VALUE "TOTAL NUMBER OF ABSENCES FOR".
010700     02 COLUMN IS 46 PIC X(9)
010800        SOURCE MONTHNAME OF MONTH-2(SAVED-MONTH).
010900     02  COLUMN 57 PIC XXX VALUE "WAS".
011000     02  TOT      COLUMN 61 PIC 999        SUM NO-ABS.
011100 01  TYPE PAGE FOOTING LINE 54.
011200     02  COLUMN 59   PICTURE X(12) VALUE "REPORT-PAGE-".
011300     02  COLUMN 71   PICTURE 99 SOURCE PAGE-COUNTER.
011400 01  TYPE REPORT FOOTING.
011500     02  LINE PLUS 1 COLUMN 32 PICTURE A(13)
011600         VALUE "END OF REPORT".
```

(1) (2) (3) (4) (5) (6) (7) (8)

```
011700 PROCEDURE DIVISION.
011800 DECLARATIVES.
011900 PAGE-HEAD-RTN SECTION.
012000     USE BEFORE REPORTING PAGE-HEAD.
012100 TEST-CONT.
012200     IF MONTH = SAVED-MONTH MOVE "(CONTINUED)" TO CONTINUED
012300         ELSE MOVE SPACES TO CONTINUED
012400             MOVE MONTH TO SAVED-MONTH.
012500 END DECLARATIVES.
012600 SORTING SECTION.
012700 SORTER. SORT PENNI ON ASCENDING KEY
012800         MONTH, DAYY, GRADE, ROOM, STUDENT
012900         USING INFILE OUTPUT PROCEDURE REPORTER.
013000 END-OF-THE-SORT.      STOP RUN.
013100 REPORTER SECTION.
013200 INITIATE-REPORT. OPEN OUTPUT REPORTFILE.
013300                  INITIATE ABS-REPORT.
013400 UNWIND-THE-SORT.
013500     RETURN PENNI  AT END
013600         TERMINATE ABS-REPORT  CLOSE REPORTFILE   ELSE
013700     GENERATE DETAIL-LINE  GO TO UNWIND-THE-SORT.
```

12. DATA MANAGEMENT

The interface to DMSII is via extensions to the standard COBOL compiler. The extensions affect non-data-base users of COBOL only in that new statements have been added to the reserved word list. No special compiler or extra control cards are required for the data base user. The extensions conform very closely to the familiar characteristics of COBOL, as can be readily seen by examining the sample programs provided in the B 7000/B 6000 HOST Language Reference Manual. Form No. 5001498.

13. COBOL COMPILER

GENERAL DESCRIPTION

This section describes the B 7000/B 6000 COBOL compiler and its operation with the MCP on the B 7000/B 6000 information processing system.

The language acceptable as input to this compiler is based on the CODASYL Journal of Development - 1968 and USASI X3.23-1968 plus the extensions to COBOL indicated in Section 1.

INPUT

Punched cards, disk, or magnetic tape may be specified as source-language input media for this compiler. On the basis of sequence numbers, the compiler has the capability of merging inputs from punched cards and either tape or disk.

A library of symbolic images may be held on disk for inclusion in the source program. The images are included in the source program by means of the COPY statement or a $ FROM.

Source-language input is handled by means of various compiler input files. The primary compiler input file is a card file with the internal name CARD. This file is required for a compilation and is normally a card reader file, but may be label-equated to another file assigned to a different device. The CARD file may be an ASCII or EBCDIC-coded file (with 14-word records) or a BCL-coded file (with 10-word records). It may be either blocked (with any block size) or unblocked as desired.

The secondary compiler input file is a disk file with the internal name TAPE. This file is optional and is normally a serial disk file; however, it may be label-equated to another file assigned to a different device. The TAPE file may be an ASCII or EBCDIC-coded file (with 14-word records) or a BCL-coded file (with 10-word records). It may be either blocked (with any block size) or unblocked as desired. This file is accepted as compiler input when the MERGE compiler option is set. The card images in this file are merged with the card images in the primary input file (CARD) according to sequence number. When duplicate sequence numbers are found on two such card images, the card from the secondary file is replaced by the card from the primary file.

The compiler may also access permanent library files as input thru use of the FROM compiler control card and the COPY statement. The files thus accessed may be either EBCDIC-coded (with 14-word records) or BCL-coded (with 10-word records); and may be either blocked (with any block size) or unblocked as desired. The internal name of a file accessed by a FROM card is SAVEPERMIN, and the TITLE of the file is specified on the FROM card and cannot be label-equated. The internal name of a file accessed by the COPY clause is LIBRARY, and the TITLE of the file is specified in the COPY clause or by label equation (see Section 8, the COBOL library).

OUTPUT

The COBOL compiler processes the given input data to produce optional output files. These output files may consist of an updated source-language file, a printer listing, an object code file, and source-language library files.

An updated source-language file, acceptable as input to the COBOL compiler, can be generated by this compiler. The updated file may be created on disk or magnetic tape by specifying the NEW compiler control option and a label-equation card. This file is assigned to serial disk unless another device is indicated on the FILE control card. This source-language file will be EBCDIC-coded, with 14-word records in 420-word blocks, and have the internal name NEWTAPE. A SAVEFACTOR of 99 and a TITLE of COBOL/IMAGE are assigned to this file. The file will be created only if the compiler option NEW is set, and it will contain all card images input during the compilation. The sequence numbers on these card images may be updated by employing the SEQ compiler option.

The output listing is an optional print file that is generated by the compiler when the compiler options LIST, or LIST1 are set. The files have the internal names LINE and ERRORFILE and contain the following items:

a. Input source-language card images.
b. Code segmentation information.
c. Program stack sizes.
d. Diagnostic messages.
e. Elapsed and processor compilation times.
f. Number of input card images scanned.
g. Sequence number of last error detected in input.
h. Number of syntax errors detected in input.
i. Estimated core memory requirement in words.
j. Generated object code (if option CODE is also set).
k. Stack address assignments (if option STACK is also set).

If the option SPEC is reset, this listing also includes a display of elementary data items being referenced by program instructions employing the CORRESPONDING expression.

The generated code is written on the disk but not saved if the compilation is specified for syntax check only or one or more syntax errors are found.

LIBRARY CREATION

Source-language libraries may be created by the compiler (by the use of SAVE compiler option) or by a program other than the compiler.

DEBUGGING AND DIAGNOSTIC FACILITIES

Diagnostic aids for programs are invoked by including appropriate control cards ($-cards) within the input to the compiler.

Error and warning messages are automatically produced (appendix F) on the output listing following the line which contains an error. The error message is followed by the syntactic item being examined at the time the error is determined. The total number of errors appears in the summary of compilation statistics at the end of the listing. When no listing is requested, only the syntax errors are listed. Warning messages inform the programmer of possibly erroneous conditions which do not prevent a code file from being produced by the compiler. For example, incorrect sequence numbers will cause a warning message unless the option SPEC is set. When set, SPEC suppresses the printing of warning messages.

Understanding the code generated by the compiler can assist the programmer in writing efficient, precise COBOL statements. The option CODE produces a list of operator mnemonics and their hexadecimal equivalents, any operands used, and the macros in the compiler producing the code.

COMPATIBILITY

CODASYL COBOL-68

Source-language programs written according to the CODASYL COBOL-68 specifications are suitable for compilation by this COBOL compiler without the need for filter or translation.

American National Standard COBOL

Source-language programs written according to the specifications for American National Standard COBOL as contained in the publication X3.23-1968 or 1974, are suitable for compilation by this COBOL compiler without the need for filter or translation when the USASI or ANSI74 system options are set. Such compatibility is subject to qualification of implementor-defined aspects, hardware-dependent facilities and areas where two or more well-recognized interpretations exist of the standard.

a. The following implementor-defined aspects of ANSI COBOL may be implemented on other systems in a manner not compatible with the syntax and/or semantics of this compiler but do not imply non-conformance with the standard:

- Hardware-name in the SPECIAL-NAMES and FILE-CONTROL paragraphs.
- The VALUE OF clause of the file description entry.
- The internal representation of data items for which USAGE IS INDEX or USAGE IS COMPUTATIONAL has been specified. No automatic alignment is provided for data items for which USAGE IS COMPUTATIONAL is specified.
- When SYNCHRONIZED is specified, implicit FILLER positions may be generated, thus affecting the size of the group and redefining items.
- The collating sequence of non-numeric items. See Appendix G for the collating sequences recognized by this compiler.
- The representation and placement of operational signs when PICTURE character S is specified for a data item are discussed in Section 6.
- The ENTER statement.

b. Some facilities of ANSI COBOL as specified in the publications X3.23-1968 and 1974 are specified in a manner which does not lead to consistent interpretation by all implementors of ANSI COBOL. No implication of non-conformance exists as a result of the interpretation adopted by the B 7000/B 6000 compiler. These facilities and the interpretation taken by B 7000/B 6000 COBOL are as follows:

- When a signed numeric integer data item is moved to an alphanumeric data item, the sign is not moved (1968 standard only).
- When a signed numeric data item is compared to a non-numeric data item, the sign, if any, participates in the comparison.
- When a class test is performed on an unsigned numeric data item or on an alphanumeric data item, and the data item tested contains an operational sign, the result of the class test will be NOT NUMERIC.
- In a WRITE statement for a printer file, when no ADVANCING phrase is specified, BEFORE ADVANCING 1 is assumed (1968 standard only).

B 5500/B 5700 COBOL-61

Source-language programs written in B 5500/B 5700 COBOL-61 are acceptable as input to the COBOL compiler after translation by the B 7000/B 6000 COBOL filter programs, with the following exceptions:

a. Data communications I/O.
b. Inter-program communication.
c. The collating sequence of the BCL character set is not compatible.
d. Options 4 and 5 of the MOVE statement.

Other Compilers

Source-language programs written for COBOL compilers of other systems may be filtered to B 7000/B 6000 COBOL by the Burroughs Conversion Aids. Alternatively, an appropriate system option may be set and the B 7000/B 6000 will flag those statements requiring modification.

COMPILER CONTROL CARDS

For program compilation, the only required cards are a COMPILE card, a DATA, EBCDIC or BCL card, the source program cards, and an END card, in that order. For program execution, the only required cards are a RUN or EXECUTE card; a DATA, EBCDIC or BCL card if input data is to be read from cards; and an END card.

The COMPILE card specifies the operation (COMPILE) to be performed, the name to be given to the object code file, and MCP options. The format of the COMPILE card is:

COMPILE program-name WITH COBOL [SYNTAX / LIBRARY / GO]

Program-name may consist of 1 to 14 names separated by slashes. Each name may contain not more than 17 alphabetic (A thru Z) and/or numeric (0 thru 9) characters. If the MCP option GO is specified or an MCP option is not specified, the code file is executed if no syntax errors are found but the code file is not saved. If the MCP option LIBRARY is specified, the code file is saved if no syntax errors are found during compilation. If the MCP option SYNTAX is specified, the program is compiled and no code file is made.

The DATA, EBCDIC or BCL card specifies the character set in which the input card file is punched. DATA and EBCDIC are synonyms.

The END card signifies the end of the input card file. Any other control card will also signify the end of the input card file and may be used in place of the END card.

Additionally, a FILE card may be used to specify that a substitute file is to be used. This card has the format:

[COBOL] FILE file-name (attribute replacements)

The word COBOL specifies that a compiler file is to be affected. When COBOL is not specified, a file within the object program is affected. (See Work Flow Language (WFL) Reference Manual, Form 5001555 for complete information on this and other control cards.)

Typical decks for program compilation and execution together with explanations of the cards used are presented in the following pages. When <I> appears as the first character of a control card, the <I> represents an invalid character (such as the combination of a 1, 2 and 3 punch) punched in column 1. Two or more control cards may be punched into one card. One group of control information is separated from the next by a semicolon instead of the invalid character when cards are combined.

COMPILE FOR SYNTAX (Card input only)

```
<I> COMPILE PROG/ONE COBOL SYNTAX
<I> EBCDIC
    Source card file
<I> END
```

COMPILE FOR SYNTAX (Card and disk input)

```
<I> COMPILE PROG/TWO COBOL SYNTAX
<I> COBOL FILE TAPE (TITLE=TWO/SOURCE)
<I> EBCDIC
$ SET MERGE
    Source card file
<I> END
```

COMPILE FOR LIBRARY (Card and tape input)

```
<I> COMPILE PROG/THREE COBOL LIBRARY
<I> COBOL FILE TAPE (TITLE=THREE,KIND=TAPE)
<I> COBOL FILE NEWTAPE(TITLE=3D)
<I> FILE MSTR(KIND=DISK)
<I> BCL
$ LIST MERGE SINGLE NEW
    Source card file
<I> END
```

COMPILE TO LIBRARY AND GO (Input on disk)

```
<I> COMPILE PROG/FOUR COBOL LIBRARY GO
<I> COBOL FILE CARD (TITLE=FOUR,KIND=DISK)
<I> END
```

COMPILE AND GO (Input on tape, patches on disk)

```
<I> COMPILE PROG/FIVE COBOL GO
<I> COBOL FILE TAPE (KIND=TAPE,TITLE=IN/FIVE)
<I> COBOL FILE CARD (KIND=DISK,TITLE=P/FIVE)
<I> EBCDIC INFILE
    Data required for execution
<I> END
```

```
COMPILE AND GO (Input on card)

     <I> COMPILE PROG/SIX COBOL;EBCDIC
         Source card file
     <I> BCL CDFILE
         Data required for execution
     <I> EBCDIC XFILL
         Second card data file required for execution
     <I> END

EXECUTION

     <I> RUN PROG/THREE
     <I> FILE MSTR(KIND=DISK,TITLE=MSTR2/PAY)
     <I> DATA DATA/DECK
         Data required for execution
     <I> END


     <I> EXECUTE PROG/FOUR; END
```

Various options are available during compilation. These compile-time options are activated by $ control cards.

$ control cards must have the $ symbol punched in either column 1 (if sequence numbering is not needed) or in column 7 or 8. The options to be manipulated are specified following the $, with one or more spaces following each option. No option may continue past column 72 of a $ card. A period in a dollar card causes the remainder of the card to be ignored.

$ control cards may be interspersed at any point within the source-language inputs $ control cards will not appear in the source-language output or in SAVEd input unless the $ symbol is in column 8 instead of column 7 or 1.

There are five types of operands which may appear on a $ control card:

a. Option action indicators.
b. Source image selection.
c. Compiler-directing options.
d. User options.
e. System compatibility options.

Option Action Indicators

Option action indicators preceding a list of options are used to set, reset, or recall the last setting of each settable option in the list.

Associated with each of the settable options is a register which reflects the last 47 settings of the option. The options FROM, PAGE, AREACLASS, LEVEL, and LIMIT are considered non-settable since they are not Boolean in nature. The presence of any preceding option action indicator is ignored when processing non-settable options.

The option action indicators are:

a. SET. The current setting of each option specified is saved and each of the options is turned on.

b. RESET. The current setting of each option specified is saved and each of the options is turned off.

c. POP. The current setting of each option specified is discarded and each of the options is set to its prior setting. When there is no prior setting, a POP will leave the option turned off.

Initial States of Settable Options

When the compiler is not initiated by way of CANDE, the initial state of the LIST, B6700, and LIBDOLLAR options is on; all other options are off. When the compiler is initiated by way of CANDE, the initial states of the LINEINFO, B6700, FREE, and LIBDOLLAR options is on; all other options are off.

Initialization of options occurs when a dollar card appears containing a settable option that is not preceded by an option action indicator. For example, the following $ card does not contain an option action indicator prior to the first settable option:

```
$ LIST  STACK
```

Note that a dollar card containing only a non-settable option does not cause dollar option initialization. For example:

```
$ LEVEL 3
```

Initialization of options causes all options to be returned to their initial states and then setting the options appearing on the dollar card. The state of the following options cannot be changed after the occurrence of the IDENTIFICATION DIVISION header in the source input: FREE, STATISTICS, INTRINSICS, LINEINFO, ANALYZE, and OPTIMIZE. These options are also not changed if option initialization occurs subsequent to the occurrence of the IDENTIFICATION DIVISION header. All prior states of an option are discarded when it is returned to its initial state.

Source Image Selection

MERGE

MERGE causes the primary input file (CARD) to be merged with the secondary input file (TAPE). If matching sequence numbers occur, the secondary input is discarded. When MERGE is not set, only records from the primary input file and the SAVEd input may be used.

NEW

NEW causes a new source-language file to be created for use later as secondary input.

SAVE

SAVE causes SAVEd input to be created for inclusion in place of a $ card specifying FROM. A SAVE may create a permanent or temporary file.

For creation of a permanent file, the syntax of the SAVE is as follows:

$ SET SAVE file-title

The SAVEd input identified by file-title becomes a permanent disk file available to any compilation. The file-title must be enclosed in quotes.

The syntax for recalling records from a permanent SAVEd input file is:

$ FROM file-title [integer-1 [THRU integer-2]]

The file-title must be identical to the file-title specified on the SAVE card.

Integer-1 specifies the beginning sequence number in the SAVE input and integer-2 specifies the ending sequence number. When integer-1 and integer-2 are not specified, all records from the SAVEd input will be included. If integer-1 is specified, integer-2 may be specified to indicate that inclusion is to end with a record which contains the sequence number specified by integer-2. When integer-2 is not specified, inclusion continues thru the end of the SAVEd input. A hyphen may be used in place of THRU.

The syntax for creating temporary SAVEd input is:

$ SET SAVE

The syntax for recalling temporary SAVEd input is

$ FROM integer-1

Integer-1 must be the sequence number of a SAVE which did not specify a file-title.

The SAVE option may be reset by a $ RESET SAVE, a $ POP SAVE, or be implicitly reset by a $ SET SAVE, a $ FROM or by initialization of dollar options. The SAVE operation will combine together a group of source image records to be used as SAVEd input. This combination of records continues until SAVE is explicitly or implicitly reset. If SAVE is implicitly reset by a $ SET SAVE, one group of records is terminated and a new group is initiated.

POPping SAVE functions exactly as "RESET" in that the option cannot be POPped to its prior value.

During a SAVE operation, the source image records will not be compiled but will instead be written to a permanent or temporary SAVEd INPUT file.

A $ SAVE or $ FROM will cause a syntax error if either one originates in a library file.

Figure 13-1 is an example of SAVE and FROM.

```
          .
          .
          .
200100    $  SET SAVE "A/SAVED/INPUT"
           01   ABC   SZ 3000.
200200    $  SET SAVE
           01   XYZ.
                05 A  SZ 10.
                05 B  SZ 10.
200300    $  RESET SAVE
           01   XABC COPY "A/SAVED/INPUT".
200400    $   FROM "A/SAVED/INPUT"
200500    $   FROM  200200
          .
          .
          .
```

Figure 13-1. Example of SAVE and FROM

Compiler—Directing Options

ANALYZE

ANALYZE, when set, will produce a listing of conditions which will adversely affect the programs demands upon system resources. The items which are flagged are not necessarily bad programming practices, but are instead items which could be handled in a more efficient manner. The ANALYZE option must be set prior to reading the division header for IDENTIFICATION DIVISION. The output listing produced when ANALYZE is set will appear at the end of the program listing, giving either the sequence number of the inefficient statement or the data-name or procedure-name which is involved in the inefficiency. The messages produced by ANALYZE and the reason why the message is produced is listed in Appendix F.

AREACLASS

An integer less than 256 must be specified following the option. This integer value will be used as the value of the AREACLASS attribute of the object code file. Any option action indicator is ignored.

CHECK

This option causes the compiler to print warning messages for sequence errors in the source-language input. (See NEWSEQERR, SEQ, SEQERR and SPEC.)

CODE

This option causes the generated object code to be listed and the option STACK to be automatically active. If LIST1 is set, pass-1 macros will be listed on the pass-1 listing. This option is not active except when LIST is set.

COMP

This option when reset, causes all 77-level COMP items to be placed in the stack in the same way as COMP-1 data items and results in faster access to these items. Setting this option causes all 77-level COMP items to be gathered together and placed in an array.

DEBUG

This option causes pseudo-stack, literal stack, and boolean linking stack locations to be printed. This option is intended to facilitate compiler development, and its function may change without notice. This option is not active except when LIST is set.

ERRLIST

When set, the ERRLIST option causes the listing of syntax errors to be written to the compiler file ERRORFILE. When the compiler is called from CANDE, option ERRLIST is automatically set and the compiler file ERRORFILE is automatically equated to the remote device involved.

FREE

This option when set, removes most of the margin restrictions required by COBOL. Division-names, section-names, and paragraph-names must start in columns 7, 8, 9, 10 or 11. All other constructs may start any place after the sequence number field. If column 7 (or the first input character when entering data thru CANDE) is $, *, /, or hyphen, then that character is interpreted as indicating a dollar control card, comments card, page eject, or continuation card respectively. This option is set automatically when the compiler is called by CANDE. If the FREE option is to be set, it must be set prior to the first source image and may not appear on any $ CARD after the first source image. Initialization of $ options does not affect FREE.

GLOBAL

This option causes all data items in WORKING-STORAGE to be global except those for which LOCAL or OWN are specifically declared. OWN and GLOBAL $ options may not both be set. The GLOBAL option is ignored unless the compilation is at level 3 or higher.

INFO

This option causes INFO, XINFO and DICT entries to be listed. INFO is intended to facilitate compiler development, and its function may change without notice.

INTRINSICS

Programs compiled at level 2 may be bound into the intrinsic file if they are compiled with the INTRINSICS option set. This type of intrinsic may be called by a program written in any language, as an untyped procedure. The GLOBAL phrase is not permitted when INTRINSICS is set.

LEVEL

This option is not settable. The word LEVEL must be followed by an integer greater than 1 and less than 31. This option controls the lexicographical level at which the compilation is to occur. A value of 2 will be assumed when no LEVEL is stated. The LEVEL must appear prior to the IDENTIFICATION DIVISION.

LIBDOLLAR

When this option is not set, $ cards read in from a library are ignored. This option is automatically set at beginning of compilation and on $ card initialization. LIBDOLLAR is ignored if it appears on a $ card which was input from a library.

LIMIT

This option is not settable. The word LIMIT must be followed by an integer value. When the number of syntax errors equals or exceeds the integer value, the compiler will terminate compilation. No limit is assumed when LIMIT is not specified except when compilation is called for by CANDE. CANDE sets LIMIT automatically to 10.

LINEINFO

This option causes source-language sequence numbers to be saved in the code file so that abnormal terminations of the object program may be identified on the SPO, in the log, and in memory dumps by the sequence number rather than by an address couple. Setting of this option requires a significantly larger amount of disk space for storage of the object program but will not affect memory requirements or usage until an abnormal termination occurs. This option is set automatically by CANDE.

LIST

This option causes the source-language to be listed during pass 2. When LIST is not set, only syntax error messages are printed. LIST is automatically reset for compilations requested by CANDE and automatically set otherwise.

LISTDELETED

This option causes source images which are replaced or deleted to be printed on the output listing.

LISTP

This option causes all source images from the primary source input (card file) to be printed on the output listing, even though the list option is reset.

LIST1

When this option is set, the compiler lists the source-language during pass 1. If, in addition, CODE is set, the pass-1 macros will also be listed.

NEWID

This option causes the contents of columns 73-80 of the source image to be replaced by a non-numeric literal. The literal must be from one to eight characters in length and need not appear in conjunction with NEWID or even be on the same $ card. Setting NEWID simply activates the replacement process. This option may be reset (via a POP or RESET or $ option initialization) to deactivate the replacement process.

NEWSEQERR

This option causes source-language records being written to NEWTAPE to be sequence checked and warning messages to be printed for out-of-sequence records. If sequence errors do occur, file NEWTAPE is not locked (and thus may be lost). This type of error will not have any effect on the successful compilation of the program.

OFFSET

This option presents offset indications for all contiguous data items, including those in the DATA-BASE section, not having their own stack location.

The offset is printed on the same line as the card image and is generated on the assumption that data descriptions in the source program are presented in the generally-accepted method; that is, one data item description per line, beginning with a level number and ending with a period. If two or more item descriptions are on the same line, the offset shown will be the offset of the first item. If a line contains only the description of an elementary unvalued filler item having no condition names, the offset shown will be the offset of the next item, rather than the offset of the filler.

The offset is printed on the extreme right of the listing in the following format:

```
0000  (WWWW:C)
```

Where "0000" is the offset of the item in terms of its character size.

Where "WWWW" is the offset in words from the beginning of the array, and

Where "C" is the character offset from the end of the last word in terms of its character size.

All values are hexadecimal.

When an item is subordinate to a non-01-level item with an OCCURS clause, only "0000" value is shown, preceded by a "+"; this is the offset from the occurring group item. The "+" indicates that it is an incremental offset.

Example:

```
01 TAB.
   03  X  PIC  X(9).              0000(0000:0)
   03  Y  PIC  9(6)  COMP-2.      0012(0001:6)
   03  Z  PIC  8(11) COMP.        0002(0002:0)
   03  A  PIC  XXX.               0012(0003:0)
   03  B  PIC  9(11) COMP.        0015(0003:3)
   03  B  OCCURS 10.              001B(0004:3)
      05  L   PIC XX.            +0000
      05  H   PIC 99 COMP-2.     +0004
   03 W   PIC X.                  0039(0009:3)
```

NOTE:

1. The offsets of COMP-2 items are in terms of 4-bit characters.

2. The offsets of word-ALIGNed COMP items are in terms of words.

3. The offsets of character-ALIGNed COMP items are in terms of 8-bit characters.

OLDNOT

When this option is set, all occurrences of NOT will be considered logical operators. The following forms are interpreted in the following manner when the OLDNOT option is set.

	FORM	INTERPRETED AS
1.	NOT A=B or C	NOT (A=B) or A=C
2.	A > B or NOT=C or D	A > B or NOT (A=C) or A=D
3.	A NOT=B or > C	A NOT=B or A > C

NOTE: This distinction becomes important when abbreviated relation conditions are used, because, under the rules of the 1968 ANSI COBOL standard, logical NOT operators apply only (in the absence of parentheses) to the immediately following relation condition, while the implied relational operator of an abbreviated relation condition is obtained from the last explicitly stated relational operator. For example, "A NOT =B or C" which was interpreted as "A NOT=B or A=C" prior to II.7, is now interpreted as "A NOT=B or A NOT =C" because the NOT is part of the relational operator.

OMIT

When set, OMIT causes card images from both file CARD and file TAPE to be ignored. That is, these card images will not be compiled, although, they may be listed and/or included in a new source file. On the output listing these images are flagged by the word OMIT.

OPTIMIZE

This option causes generated code to be optimized for fastest execution. The setting of this option may not be altered after the first source image. OPTIMIZE is initialized to reset. With OPTIMIZE set, the following optimizations are effected:

a. Numerical comparisons involving unsigned DISPLAY items will be compared as characters rather than being converted to binary and then compared. The zone portion of each character is masked with hex "F" so that uninitialized data items will compare equal to zero.
b. Simple ADD statements involving unsigned DISPLAY items will be done as character adds with all zones masked with HEX "F" so that uninitialized data items will be considered zero.
c. Invalid index checking on subscripting is bypassed. The responsibility for ensuring the validity of any given subscript then rests with the programmer. System protection against accessing beyond the bounds of an array is still effective.

d. The characteristics of the program's PERFORM statements and ranges will be analyzed. PERFORM ranges containing less than "about" 5 or 6 statements and performed by less than "about" 5 or 6 PERFORM statements will be compiled inline at each PERFORM statement. There will be no overhead for doing the PERFORM. The numbers "5" and "6" are merely estimates in an attempt to maintain balance between an increased amount of object code and the overhead required to do a PERFORM. PERFORM ranges which are not compiled inline, but which are not executed except under control of a PERFORM statement, will have a special unconditional return mechanism produced at the end of the PERFORM range which will decrease the PERFORM overhead.

OWN — This option causes all WORKING-STORAGE data items to assume the declaration OWN except those for which LOCAL or GLOBAL is explicitly declared. The OWN and GLOBAL $ option is ignored unless the compilation is at level 3 or higher.

PAGE — This option is not settable. The appearance of PAGE on a $ card causes the pass-1 and/or pass-2 listings to skip to a new page.

SECGROUP — This option causes the normal code segmentation of the compiler to be overridden so that groups of two or more sections may become a single code segment. When this option is set, contiguous non-declarative sections which specify the same priority number on the section header will be grouped into a single code segment. When the option is not set, segmentation is controlled by section headers without consideration of priority number, and sections will not be grouped together. For purposes of SECGROUP, sections without a priority number are never combined with any other section. Sort input or output procedures will never be combined with each other or with any non-sort procedures. SECGROUP will not be effective on declarative sections. The SEGMENT-LIMIT clause is effective regardless of the setting of SECGROUP.

SEQ

This option activates the resequencing of source-language output files (i.e., NEWTAPE, pass-1 listing, and pass-2 listing). The SEQ option is normally associated with a beginning sequence number and an increment which is added for subsequent source-language images. These two integer values may be specified on a $ card prior to the $ card which activates the SEQ, on the $ card which activates the SEQ, or the default value (10) supplied automatically by the compiler may be used. As long as SEQ is set, the beginning sequence number is incremented by the increment for each source-language record, and that value is retained while SEQ is not set. Thus, setting SEQ does not restore the beginning sequence number. The beginning sequence number is initialized when an unsigned integer is found in any $ card (with or without SEQ set). The increment is initialized by specifying a plus sign (+) followed by an integer in any $ card (with or without SEQ set). The two integer values need not be specified in the same $ card nor even on the $ card specifying the SEQ option.

SEQERR

With this option, the compiler will print warning messages for sequence errors in the source-language input. At the end of a compilation which had sequence errors, a code file will not be created. Thus, sequence errors which print as warnings behave as though they were fatal when SEQERR is set. The settings of NEWSEQERR, SEQ, SPEC and CHECK have no effect on the setting of SEQERR.

SINGLE

This option causes the compiler output listing to be single spaced. SINGLE is automatically set unless the option DOUBLESPACE is set when the compiler is compiled.

SPEC

The SPEC option suppresses printing of warning messages, sequence error messages, printing of the expansion of the DMS INVOKE statement, and the printing of the list of elementary items involved in a CORRESPONDING option.

STACK

This option causes relative stack addresses and the name of the associated item to be printed on the output listing. The STACK option will become active when STACK is set or both LIST and CODE are set.

STATISTICS

It is possible to obtain statistics which reveal the characteristics of a COBOL object job. Statistics are accumulated for a program when the STATISTICS dollar option is set. This option may not be changed after the compiler has encountered the beginning of the IDENTIFICATION DIVISION. When this option is set, the compiler will include code to determine how many times each paragraph is entered and how much time is spent executing the instructions comprising each paragraph. The STATISTICS dollar option can only be set for a compilation at level 2.

Each paragraph has a unique number. This number is printed on the right-hand side of the compiler listing and corresponds to one line of output on the system summary of the statistics. This summary is written on the job's diagnostic file (the file on which program dumps appear). A simple example follows:

```
PROCEDURE DIVISION.

P1.
                                        ACTIVE CLOCK IS #0001
   OPEN INPUT CARD-FILE.
P2.
                                        ACTIVE CLOCK IS #0002
   READ CRD AT END GO TO DONE.
   IF FLD=0 THEN PERFORM ZERO-
   RECORD.
P3.
                                        ACTIVE CLOCK IS #0003
   MOVE 5 TO R-CLASS.
```

The output of the statistics summary would look like the following example:

BLOCK	FREQ	TOTAL TIME	AVG TIME
MAIN	1	0.132267	0.132267
1	1	600	600
2	1000	0.500000	500
3	366	36600	100
.	.	.	.
.	.	.	.
.	.	.	.

The column labeled BLOCK specifies the paragraph, as numbered on the source listing, for which the line of output applies. The line labeled MAIN is the time necessary to initialize user data areas and construct the stack.

The data listed under the heading FREQ reflects the number of times the paragraph was entered. Paragraphs never entered are not listed.

The data under TOTAL TIME is the total elapsed time spent processing the paragraph. The column AVG TIME is equal to the value of TOTAL TIME divided by FREQ. One should note that times printed in both of these columns without a decimal point are times in microseconds. Thus the total time spent in P1 would be 600 microseconds; but, for P2 it would be one half of a second.

A statistics summary is produced at END-OF-TASK, or when the program is DS-ed.

The times accumulated are elapsed times. Therefore, they reflect the time needed to handle presence bit interrupts, I/O interrupts, and possibly, time spent in other processes. Also, the timing does not cease upon entering the MCP (READ, WRITE, etc.) or using IPC (CALL, PROCESS, CONTINUE, etc.). The PERFORM statement, however, does stop timing and begin timing the performed paragraph, resuming the timing of the performing paragraph upon exit from the PERFORM range.

TIME	This option causes the compiler to print the normal heading and footing of the compilation listing even though the option LIST was never set.
VOID	This option causes all source-language input (primary and secondary), except $ cards, to be ignored until the option becomes not set.
VOIDT	This option causes all secondary source-language input except $ cards to be ignored until the option becomes not set. This option is not active unless the MERGE option is set.
XREF	When this option is set, a cross-reference listing will be produced of declarations and uses of all data-names, file-names, condition-names, etc. Operation of XREF is in no way dependent on the setting of LIST. By use of SET, POP, and RESET, the XREF operation may be restricted to portions of the program.
$	This option causes all $ control cards to be listed.
Integer	When an integer value (not preceded by the symbol +) appears on a $ control card, the integer value is used as the beginning sequence number for the operation of the SEQ operation. This is not a settable option.
+ Integer	When an integer value with a preceding plus symbol (with or without intervening blank spaces) appears in a $ control card, the integer value will be used as the increment for operation of the SEQ $ option. This is not a settable option.
Non-Numeric Literal	When a non-numeric literal is specified in a $ control card, the literal will be used for the replacement operation performed by the NEWID $ option.

User Defined Dollar Options

Identifiers not having the same length and first five characters as a standard dollar option will be considered to be user options.

In addition, any option, user or standard, may be set to the value of an "option expression", as is the case in ESPOL and ALGOL. The syntax for this conditional set is:

```
$ SET option = option expression
```

where

```
option expression :: =  option secondary/option expression
                        Boolean operator option secondary

option secondary ::  =  option primary/NOT option primary

option primary ::    =  option/(option expression)

option ::            =  standard option/user option

Boolean operator ::  =  AND/OR/IMP/EQV
```

In the following card deck for example, cards in Region-1 will be omitted and cards in Region-2 will not be omitted.

```
$SET TESTING KLUDGE
    .
    .
    .
$SET OMIT = TESTING AND KLUDGE
    .
    .    (Region-1)
    .
$POP OMIT
    .
    .
    .
$SET OMIT = NOT (TESTING OR KLUDGE)
    .
    .    (Region-2)
    .
$POP OMIT
    .
    .
    .
```

The special-action parameter options "AREACLASS", "FROM", "LEVEL", "LIMIT", and "PAGE" are not permitted in an option expression, since they imply no Boolean value. The normal precedence of Boolean operators applies to the evaluation of option expressions. Dollar cards having options, but no option action, will cause all options, both standard and user options, to be cleared. For example, given the user option TESTING, $ TESTING will cause the same action as described on page 13-11 for an option with no option indicator.

SYSTEM COMPATIBILITY OPTIONS

System compatibility options are provided to restrict recognition of reserved words. These options are B2500 for all models of B 2500, B 2700, N 3500, B 3700 and B 4700; B5700 for B 5500 and B 5700 COBOL-61; B6700 for B 6500, B 6700, B 7700, and B 5500/B 5700 COBOL-68; USASI for COBOL as specified in the publication USASI COBOL, X3.23-1968, S360 for IBM 360 and IBM 370; and ANSI74 for COBOL as specified in ANSI X3.23-1974.

The compiler's reserved word table includes words which are reserved on B7000/B 6000, words which are reserved on other systems compilers (but not on the B 7000/B 6000), and words which are reserved on both. Setting the appropriate system option will allow words which are reserved on one system to be used on another system as paragraph-names, data-names, etc. An example of this is the word PROCESSING which is reserved if B2500, USASI, or S360 system options are set and is available to the user as an identifier when only B6700 or B5700 are set.

A system option may be set by $ option initialization or by a $SET.

The setting of system options are not mutually exclusive; that is, setting a system option will not reset all other system options, allowing words and facilities if they are reserved or allowed under any of the systems specified. Systems options can also be reset or popped independently. Initialization of system options will occur if an attempt is made to reset all system options. The initial value of the B5700, S360 and USASI system options is reset. The initial value of the B6700 system option is set. The initial values for the ANSI74 and B2500 system options can be made to be either set or reset, depending on how the COBOL compiler is compiled. If ANSI74 or B2500 user dollar options are set when the COBOL compiler is compiled with ALGOL, the compiler will cause the initial values of the ANSI74 and B2500 system options to be set.

The reserved word list shows all reserved words known to this compiler and indicates under which system options they will be recognized as reserved words (Appendix A).

In addition to reserved word recognition, system options also affect syntax and semantics where compatability can be enhanced as follows:

USASI	ELSE is associated only with IF. A CLOSE of a multi-file tape will assume NO REWIND unless some other option is specified. The implicit definition of TALLY will be a computational item with a PICTURE of 9(5).
S360	ELSE is associated only with IF.
ANSI74	ELSE phrases are paired only with IF statements.

A. RESERVED WORDS

The following pages list all the reserved words known to the B 7000/B 6000 COBOL compiler and indicates to which system or systems they are reserved.

An X at the intersection of a reserved word and a system option setting indicates the word is reserved when that system option is set. A hyphen indicates that the reserved word will not be recognized as a reserved word when that system option is set.

RESERVED WORD	SYSTEM OPTIONS					
	B6700	B5700	B2500	USASI	S360	ANSI74
ABS	X	X	-	-	-	-
ACCEPT	X	X	X	X	X	X
ACCESS	X	X	X	X	X	X
ACTUAL	X	X	X	X	X	-
ADD	X	X	X	X	X	X
ADDRESS	X	X	-	X	X	-
ADVANCING	X	X	X	X	X	X
AFTER	X	X	X	X	X	X
ALL	X	X	X	X	X	X
ALLOW	X	-	-	-	-	-
ALPHABETIC	X	X	X	X	X	X
ALPHANUMERIC	-	X	-	-	-	-
ALSO	X	-	-	-	-	X
ALTER	X	X	X	X	X	X
ALTERNATE	X	X	X	X	X	X
AN	-	X	-	-	-	-
AND	X	X	X	X	X	X
APPLY	X	X	X	-	X	-
ARCTAN	X	X	-	-	-	-
ARE	X	X	X	X	X	X
AREA	X	X	X	X	X	X
AREAS	X	X	X	X	X	X

RESERVED WORD	SYSTEM OPTIONS					
	B6700	B5700	B2500	USASI	S360	ANSI74
AS	X	-	-	-	-	-
ASCENDING	X	X	X	X	X	X
ASCII	X	-	-	-	-	-
ASSIGN	X	X	X	X	X	X
AT	X	X	X	X	X	X
ATTACH	X	-	-	-	-	-
AUTHOR	X	X	X	X	X	X
AUXILIARY	X	X	X	-	-	-
AWAIT	X	-	-	-	-	-
BACKUP	X	X	X	X	X	-
BEFORE	X	X	X	X	X	X
BEGIN-TRANSACTION	X	-	-	-	-	-
BEGINNING	X	X	X	X	X	-
BLANK	X	X	X	X	X	X
BLOCK	X	X	X	X	X	X
BOTTOM	-	-	-	-	-	X
BY	X	X	X	X	X	X
BZ	-	X	X	-	-	-
CALL	X	-	-	-	X	X
CANCEL	X	-	-	-	X	X
CARD-PUNCH	X	X	X	X	X	-
CARD-READER	X	X	X	X	X	-
CARD-READERS	X	X	X	X	X	-
CAUSE	X	-	-	-	-	-
CD	X	-	-	-	-	X
CF	X	-	X	X	X	X
CH	X	-	X	X	X	X
CHANGE	X	-	-	-	-	-
CHANNEL	X	X	X	X	-	-
CHARACTER	X	X	-	-	-	X
CHARACTERS	X	X	X	X	X	X
CHECKPOINT	X	-	-	-	-	-
CHECKPOINT-STATUS	X	X	-	-	-	-
CLASS	-	-	-	-	-	-
CLEAR	X	-	-	-	-	-
CLOCK-UNITS	X	-	-	X	X	X

RESERVED WORD	SYSTEM OPTIONS					
	B6700	B5700	B2500	USASI	S360	ANSI74
CLOSE	X	X	X	X	X	X
CMP	-	X	X	-	-	-
CMP-1	-	X	X	-	-	-
COBOL	X	X	X	X	X	X
CODE	X	-	X	X	X	X
CODE-SET	X	-	-	-	-	X
COLLATING	X	-	-	-	-	X
COLUMN	X	-	X	X	X	X
COMMA	X	-	X	X	X	X
COMP	X	X	X	X	X	X
COMP-1	X	X	-	-	X	-
COMP-2	X	-	-	-	X	-
COMP-4	X	-	-	-	X	-
COMP-5	X	-	-	-	X	-
COMPILETIME	X	-	-	-	-	-
COMPUTATIONAL	X	X	X	X	X	X
COMPUTATIONAL-1	X	X	X	-	X	-
COMPUTATIONAL-2	X	-	-	-	X	-
COMPUTATIONAL-4	X	-	-	-	X	-
COMPUTATIONAL-5	X	-	-	-	X	-
COMPUTE	X	X	X	X	X	X
CONFIGURATION	X	X	X	X	X	X
CONSTANT	X	X	-	-	X	-
CONTAINS	X	X	X	X	X	X
CONTENT	X	-	-	-	-	-
CONTINUE	X	-	-	-	-	-
CONTROL	X	X	X	X	X	X
CONTROL-POINT	X	-	-	-	-	-
CONTROLS	X	-	-	X	X	X
CONVERSION	X	-	X	X	X	-
COPY	X	X	X	X	X	X
CORR	X	X	X	X	X	X
CORRESPONDING	X	X	X	X	X	X
COS	X	X	-	-	-	-
COUNT	X	-	-	-	-	X
CP	X	-	-	-	-	-

RESERVED WORD	SYSTEM OPTIONS					
	B6700	B5700	B2500	USASI	S360	ANSI74
CREATE	X	-	-	-	-	-
CRUNCH	X	X	-	-	-	-
CURRENCY	X	-	X	X	X	X
CURRENT-DATE	-	-	-	-	X	-
CYLINDER	X	-	X	-	-	-
DATA	X	X	X	X	X	X
DATA-BASE	X	-	-	-	-	-
DATE	X	-	X	-	-	X
DATE-COMPILED	X	X	X	X	X	X
DATE-WRITTEN	X	X	X	X	X	X
DAY	X	-	-	-	-	X
DB	X	-	-	-	-	-
DE	X	-	-	X	X	X
DEALLOCATE	X	-	-	-	-	-
DECIMAL-POINT	X	-	X	X	X	X
DECLARATIVES	X	X	X	X	X	X
DELETE	X	-	-	-	-	X
DELIMITED	X	-	-	-	-	X
DELIMITER	X	-	-	-	-	X
DEPENDING	X	X	X	X	X	X
DESCENDING	X	X	X	X	X	X
DETACH	X	-	-	-	-	-
DETAIL	X	-	-	X	X	X
DIRECT	X	-	-	-	-	-
DISABLE	X	-	-	-	-	X
DISALLOW	X	-	-	-	-	-
DISC	-	-	X	-	-	-
DISK	X	X	X	X	X	X
DISKPACK	X	X	X	X	X	X
DISKPACKS	X	X	X	X	X	-
DISPLAY	X	X	X	X	X	X
DISPLAY-UNIT	X	X	X	X	X	-
DISPLAY-1	X	-	-	-	-	-
DIV	X	X	-	-	-	-
DIVIDE	X	X	X	X	X	X
DIVIDED	-	X	-	-	-	-

RESERVED WORD	SYSTEM OPTIONS					
	B6700	B5700	B2500	USASI	S360	ANSI74
DIVISION	X	X	X	X	X	X
DMTERMINATE	X	-	-	-	-	-
DOWN	X	-	X	X	X	X
DUMP	X	X	X	X	X	-
DUPLICATE	X	-	-	-	-	-
DUPLICATES	X	-	-	-	-	X
DYNAMIC	X	-	-	-	-	X
ELSE	X	X	X	X	X	X
EMI	X	-	-	-	X	X
ENABLE	X	-	X	-	-	X
END	X	X	X	X	X	X
END-OF-PAGE	X	-	-	-	X	X
END-TRANSACTION	X	-	-	-	-	-
ENDING	X	X	X	X	X	-
ENTER	X	-	X	X	X	X
ENVIRONMENT	X	X	X	X	X	X
EOP	X	-	-	-	X	X
EQUAL	X	X	X	X	X	X
EQUALS	X	X	X	-	X	-
ERROR	X	X	X	X	X	X
EVENT	X	-	-	-	-	-
EVERY	X	X	X	X	X	X
EXAMINE	X	X	X	X	X	-
EXCEEDS	X	X	-	-	X	-
EXCEPTION	X	-	-	-	-	X
EXCHANGE	X	-	-	-	-	-
EXECUTE	X	-	-	-	-	-
EXIT	X	X	X	X	X	X
EXP	X	X	-	-	-	-
EXPONENTIATED	-	X	-	-	-	-
EXTEND	-	-	-	-	-	X
EXTERNAL	X	-	-	-	-	-
FD	X	X	X	X	X	X
FILE	X	X	X	X	X	X
FILE-CONTROL	X	X	X	X	X	X
FILE-LIMIT	X	X	X	X	X	-

RESERVED WORD	SYSTEM OPTIONS					
	B6700	B5700	B2500	USASI	S360	ANSI74
FILE-LIMITS	X	X	X	X	X	-
FILLER	X	X	X	X	X	X
FINAL	X	-	-	X	X	X
FIND	X	-	-	-	-	-
FIRST	X	X	X	X	X	X
FIRSTONE	X	-	-	-	-	-
FOOTING	X	-	-	X	X	X
FOR	X	X	X	X	X	X
FREE	X	-	-	-	-	-
FROM	X	X	X	X	X	X
GENERATE	X	-	-	X	X	X
GIVING	X	X	X	X	X	X
GLOBAL	X	-	-	-	-	-
GO	X	X	X	X	X	X
GREATER	X	X	X	X	X	X
GROUP	X	-	-	X	X	X
HEADING	X	-	-	X	X	X
HERE	X	-	-	-	-	-
HIGH-VALUE	X	X	X	X	X	X
HIGH-VALUES	X	X	X	X	X	X
I-O	X	X	X	X	X	X
I-O-CONTROL	X	X	X	X	X	X
ID	X	X	X	-	X	-
IDENTIFICATION	X	X	X	X	X	X
IF	X	X	X	X	X	X
IN	X	X	X	X	X	X
INDEX	X	-	X	X	X	X
INDEXED	X	-	X	X	X	X
INDICATE	X	-	-	X	X	X
INITIAL	X	-	-	-	-	X
INITIATE	X	-	-	X	X	X
INPUT	X	X	X	X	X	X
INPUT-OUTPUT	X	X	X	X	X	X
INQUIRY	X	-	-	-	-	-
INSERT	X	-	-	-	-	-

RESERVED WORD	SYSTEM OPTIONS					
	B6700	B5700	B2500	USASI	S360	ANSI74
INSPECT	X	-	-	-	-	X
INSTALLATION	X	X	X	X	X	X
INTERCHANGE	X	-	-	-	-	-
INTERRUPT	X	-	-	-	-	-
INTO	X	X	X	X	X	X
INVALID	X	X	X	X	X	X
INVOKE	X	-	-	-	-	-
IS	X	X	X	X	X	X
JUST	X	X	X	X	X	X
JUSTIFIED	X	X	X	X	X	X
JS	-	X	X	-	-	-
KEY	X	X	X	X	X	X
KEYBOARD	X	X	X	X	X	-
KEYS	X	X	-	X	X	-
LABEL	X	X	X	X	X	X
LAST	X	-	-	X	X	X
LD	X	-	-	-	-	-
LEADING	X	X	X	X	X	X
LEFT	X	X	X	X	X	X
LENGTH	X	-	-	-	-	X
LESS	X	X	X	X	X	X
LIBRARY	-	X	X	-	X	-
LIMIT	X	-	X	X	X	X
LIMITS	X	-	X	X	X	X
LINAGE	X	-	-	-	X	X
LINAGE-COUNTER	X	-	-	-	X	X
LINE	X	X	-	X	X	X
LINE-COUNTER	X	-	-	X	X	X
LINES	X	X	X	X	X	X
LINKAGE	X	-	-	-	X	X
LN	X	X	-	-	-	-
LOCAL	X	-	-	-	-	-
LOCAL-STORAGE	X	-	-	-	-	-
LOCK	X	X	X	X	X	X
LOCKED	X	-	-	-	-	-

RESERVED WORD	SYSTEM OPTIONS					
	B6700	B5700	B2500	USASI	S360	ANSI74
LOW-VALUE	X	X	X	X	X	X
LOW-VALUES	X	X	X	X	X	X
LOWER-BOUND	X	X	-	-	X	-
LOWER-BOUNDS	X	X	-	-	X	-
MAX	X	-	-	-	-	-
MD	-	X	-	-	-	-
MEMORY	X	X	X	X	X	X
MERGE	X	X	X	-	-	X
MESSAGE	X	-	-	-	-	X
MESSAGE-PRINTER	X	X	X	X	X	-
MIN	X	-	-	-	-	-
MINUS	-	X	-	-	-	-
MOD	X	X	X	-	-	-
MODE	X	X	X	X	X	X
MODIFY	X	-	-	-	-	-
MODULES	X	X	X	X	X	X
MONITOR	X	X	X	X	X	-
MOVE	X	X	X	X	X	X
MULTIPLE	X	X	X	X	X	X
MULTIPLE-I-O	X	-	-	-	-	-
MULTIPLIED	-	X	-	-	-	-
MULTIPLY	X	X	X	X	X	X
MYJOB	X	-	-	-	-	-
MYSELF	X	-	-	-	-	-
NATIVE	X	-	-	-	-	X
NEGATIVE	X	X	X	X	X	X
NEXT	X	X	X	X	X	X
NO	X	X	X	X	X	X
NON-STANDARD	X	X	X	-	-	-
NOT	X	X	X	X	X	X
NOTE	-	X	X	X	X	-
NULL	X	-	-	-	-	-
NUMBER	X	-	-	X	X	X
NUMERIC	X	X	X	X	X	X
OBJECT-COMPUTER	X	X	X	X	X	X

RESERVED WORD	SYSTEM OPTIONS					
	B6700	B5700	B2500	USASI	S360	ANSI74
OBJECT-PROGRAM	X	X	X	-	-	-
OC	X	X	X	-	-	-
OCCURS	X	X	X	X	X	X
OF	X	X	X	X	X	X
OFF	X	X	X	X	X	X
OMITTED	X	X	X	X	X	X
ON	X	X	X	X	X	X
ONES	X	-	-	-	-	-
OPEN	X	X	X	X	X	X
OPTIONAL	X	X	X	X	X	X
O-I	X	-	X	-	-	-
OR	X	X	X	X	X	X
ORGANIZATION	X	-	-	-	-	X
OTHERWISE	X	X	X	-	X	-
OUTPUT	X	X	X	X	X	X
OVERFLOW	X	-	X	-	X	X
OWN	X	-	-	-	-	-
PAGE	X	-	X	X	X	X
PAGE-COUNTER	X	-	-	X	X	X
PAPER-TAPE-PUNCH	X	X	X	X	X	-
PAPER-TAPE-READER	X	X	X	X	X	-
PC	-	X	X	-	-	-
PERFORM	X	X	X	X	X	X
PETAPE	X	X	X	X	X	-
PF	X	-	-	X	X	X
PH	X	-	-	X	X	X
PIC	X	X	X	X	X	X
PICTURE	X	X	X	X	X	X
PLUS	X	X	-	X	X	X
POINTER	X	-	-	-	-	X
POSITION	X	X	X	X	X	X
POSITIVE	X	X	X	X	X	X
PREPARED	X	X	-	-	X	-
PRINTER	X	X	X	X	X	X
PRINTERS	X	X	X	X	X	-

RESERVED WORD	SYSTEM OPTIONS					
	B6700	B5700	B2500	USASI	S360	ANSI74
PRIOR	X	-	-	-	-	-
PRIORITY	X	X	X	-	X	-
PROCEDURE	X	X	X	X	X	X
PROCEED	X	X	X	X	X	X
PROCESS	X	-	-	-	X	-
PROCESSING	-	-	X	X	X	-
PROGRAM	X	-	-	-	X	X
PROGRAM-ID	X	X	X	X	X	X
PT-PUNCH	-	-	X	-	-	-
PT-READER	-	-	X	-	-	-
PUNCH	X	X	X	X	X	X
PURGE	X	X	X	X	X	-
QUEUE	X	-	-	-	-	X
QUOTE	X	X	X	X	X	X
QUOTES	X	X	X	X	X	X
RANDOM	X	X	X	X	X	X
RANGE	X	X	-	-	X	-
RD	X	-	-	X	X	X
READ	X	X	X	X	X	X
READER	X	X	X	X	X	X
READER-SORTER	X	-	-	-	-	-
READERS	X	X	X	X	X	-
RECEIVE	X	-	-	-	-	X
RECEIVED	X	-	-	-	-	-
RECORD	X	X	X	X	X	X
RECORDING	X	X	X	-	X	-
RECORDS	X	X	X	X	X	X
RECREATE	X	-	-	-	-	-
REDEFINES	X	X	X	X	X	X
REEL	X	X	X	X	X	X
REEL-NUMBER	X	X	-	-	-	-
REF	X	-	-	-	-	-
REFERENCE	X	-	-	-	-	-
RELEASE	X	X	X	X	X	X

RESERVED WORD	SYSTEM OPTIONS					
	B6700	B5700	B2500	USASI	S360	ANSI74
REMAINDER	X	X	X	X	-	X
REMARKS	-	X	X	X	X	-
REMOTE	X	X	X	X	X	X
REMOVAL	-	-	-	-	-	X
REMOVE	X	-	-	-	-	-
RENAMES	X	X	X	X	X	X
REPLACING	X	X	X	X	X	X
REPORT	X	-	-	X	X	X
REPORTING	X	-	-	X	X	X
REPORTS	X	-	-	X	X	X
RERUN	X	X	X	X	X	X
RESERVE	X	X	X	X	X	X
RESET	X	-	-	X	X	X
RETURN	X	X	X	X	X	X
REVERSED	X	X	X	X	X	X
REWIND	X	X	X	X	X	X
REWRITE	X	-	-	-	-	X
RF	X	-	-	X	X	X
RH	X	-	-	X	X	X
RIGHT	X	X	X	X	X	X
ROUNDED	X	X	X	X	X	X
RUN	X	X	X	X	X	X
SAME	X	X	X	X	X	X
SAVE	X	-	X	-	-	-
SAVE-FACTOR	X	X	X	X	X	-
SD	X	X	X	X	X	X
SEARCH	X	-	X	X	X	X
SECTION	X	X	X	X	X	X
SECURITY	X	X	X	X	X	X
SEEK	X	X	X	X	X	-
SEGMENT	X	X	-	-	-	X
SEGMENT-LIMIT	X	X	X	X	X	X
SELECT	X	X	X	X	X	X
SEND	X	-	X	-	-	X

RESERVED WORD	SYSTEM OPTIONS					
	B6700	B5700	B2500	USASI	S360	ANSI74
SENTENCE	X	X	X	X	X	X
SEPARATE	X	-	-	-	X	X
SEQUENCE	X	-	-	-	-	X
SEQUENTIAL	X	X	X	X	X	X
SET	X	-	X	X	X	X
SIGN	X	X	X	X	X	X
SIGNED	-	X	X	-	-	-
SIN	X	X	-	-	-	-
SINGLE	X	-	X	-	-	-
SIZE	X	X	X	X	X	X
SN	-	X	-	-	-	-
SORT	X	X	X	X	X	X
SORT-TAPE	X	X	X	X	X	-
SORT-TAPES	X	X	X	X	X	-
SOURCE	X	-	-	X	X	X
SOURCE-COMPUTER	X	X	X	X	X	X
SPACE	X	X	X	X	X	X
SPACES	X	X	X	X	X	X
SPECIAL-NAMES	X	X	X	X	X	X
SPO	X	X	X	X	X	-
SQRT	X	X	-	-	-	-
STANDARD	X	X	X	X	X	X
STANDARD-1	X	-	-	-	-	X
START	X	-	-	-	-	X
STATUS	X	X	-	X	X	X
STOP	X	X	X	X	X	X
STORE	X	-	X	-	-	-
STRING	X	-	-	-	-	X
SUBTRACT	X	X	X	X	X	X
SUM	X	-	-	X	X	X
SUPERVISOR	X	X	X	-	X	-
SUSPEND	X	-	-	-	X	-
SY	-	X	X	-	-	-
SYMBOLIC	X	X	X	-	X	X
SYNC	X	X	X	X	X	X

RESERVED WORD	SYSTEM OPTIONS					
	B6700	B5700	B2500	USASI	S360	ANSI74
SYNCHRONIZED	X	X	X	X	X	X
SYSTEM	X	-	-	-	-	-
SZ	X	X	-	-	-	-
TABLE	X	-	-	-	-	X
TALLY	X	X	X	X	X	-
TALLYING	X	X	X	X	X	X
TAPE	X	X	X	X	X	X
TAPE-PE	-	-	X	-	-	-
TAPE-7	-	-	X	-	-	-
TAPE-9	-	-	X	-	-	-
TAPES	X	X	X	X	X	-
TAPE7	X	X	X	X	X	-
TAPE9	X	X	X	X	X	-
TERMINAL	X	-	-	-	-	X
TERMINATE	X	-	-	X	X	X
TEXT	X	-	-	--	-	X
THAN	X	X	X	X	X	X
THEN	-	X	X	-	X	-
THROUGH	X	X	X	X	X	X
THRU	X	X	X	X	X	X
TIME	X	X	X	-	-	X
TIMES	X	X	X	X	X	X
TO	X	X	X	X	X	X
TODAYS-DATE	X	X	X	-	-	-
TOP	-	-	-	-	-	X
TRAILING	X	-	-	-	X	X
TYPE	X	-	-	X	X	X
UNEQUAL	X	X	X	-	X	-
UNIT	X	-	-	X	X	X
UNLOCK	X	X	-	-	-	-
UNSTRING	X	-	-	-	-	X
UNTIL	X	X	X	X	X	X
UP	X	-	X	X	X	X
UPDATE	X	-	-	-	X	-
UPON	X	X	X	X	X	X

RESERVED WORD	SYSTEM OPTIONS					
	B6700	B5700	B2500	USASI	S360	ANSI74
UPPER-BOUND	X	X	-	-	X	-
UPPER-BOUNDS	X	X	-	-	X	-
USAGE	X	X	X	X	X	X
USE	X	X	X	X	X	X
USING	X	X	X	X	X	X
VA	X	X	X	-	-	-
VALUE	X	X	X	X	X	X
VALUES	X	X	-	X	X	X
VARYING	X	X	X	X	X	X
WAIT	X	-	X	-	-	-
WHEN	X	X	X	X	X	X
WHERE	X	-	-	-	-	-
WITH	X	X	X	X	X	X
WORDS	X	X	X	X	X	X
WORKING-STORAGE	X	X	X	X	X	X
WRITE	X	X	X	X	X	X
ZERO	X	X	X	X	X	X
ZEROES	X	X	X	X	X	X
ZEROS	X	X	X	X	X	X

B. ANSI74 IMPLEMENTATIONS

INTRODUCTION

This appendix defines the B 7000 / B 6000 COBOL language facilities for handling specific features of the ANSI 74 COBOL language standard.

The COBOL specifications contained in the ANSI 74 standard are based on a Nucleus and eleven functional processing modules. Four modules out of the ANSI 74 standard have been implemented in B 7000 / B 6000 COBOL; they are the Indexed I/O, Sequential I/O, Table Handling, and Nucleus modules.

Certain facilities are relatively minor additions or modifications to existing language elements. These facilities are noted in this appendix; however, a complete description of these facilities is included in the appropriate main body sections of this manual.

Other facilities are completely independent of existing language elements and will be described fully in this appendix.

All ANSI 74 facilities have been implemented under the "ANSI74" system dollar option. "ANSI74" can either be set when compiling the COBOL compiler or set at the time a COBOL program is compiled. Refer to the SYSTEM COMPATIBILITY OPTIONS in section 13 for a complete description of the "ANSI74" system dollar option characteristics.

INDEXED I-O

The following ANSI 74 Indexed I-O features are described in detail on the following pages.

INDEXED I-O is implemented in the B 7000 / B 6000 series of computer systems by means of the ISAM intrinsics, and allows all constructs of the full Level 2 ANSI 74 specification with the exception of alternate record keys.

The Indexed I/O module provides a capability to access records of a mass storage file in either a random or sequential manner. Each record in an indexed file is uniquely identified by the value of one key within that record.

The FILE-CONTROL paragraph format for INDEXED I-O is:

```
INPUT-OUTPUT SECTION
FILE-CONTROL.
SELECT file-name
     ASSIGN TO [integer-1  [* integer-2]]  {DISK     }
                                           {DISKPACK }
      [; RESERVE  integer-3  [AREA ]]
                             [AREAS]
       ; ORGANIZATION IS INDEXED
      [; ACCESS MODE IS {SEQUENTIAL}]
                        {RANDOM    }
                        {DYNAMIC   }
       ; RECORD KEY IS data-name-1
      [; FILE STATUS IS data-name-2]
```

The SELECT, ASSIGN and RESERVE clauses have the same specifications as described previously in the section dealing with the FILE-CONTROL paragraph.

The ORGANIZATION clause specifies the logical structure of a file. The file organization is established at the time a file is created and cannot subsequently be changed.

NOTE: Indexed files do not have the same default values for the attributes "AREAS" (integer-1) and AREASIZE (integer-2), as normal system files. Therefore, if omitted, "AREAS" and "AREASIZE" will be set to 1. Also, the system attribute "FLEXIBLE" is not valid for indexed files being created. This means that when creating an indexed file, "AREAS" and "AREASIZE" must be made large enough to accommodate all records. After a file has been created; however, the "FLEXIBLE" attribute is valid and additional records may be added.

When the ACCESS mode is SEQUENTIAL, records in the file are accessed in ascending record key values.

When the ACCESS mode is RANDOM, the value of the record key data item indicates the record to be accessed.

When the ACCESS mode is DYNAMIC, records in the file may be accessed sequentially and/or randomly.

When a file is opened in the output mode, records must be written with ascending keys, regardless of the access mode.

The RECORD KEY clause specifies the record key for the file. The values of the record key must be unique among records of the files. The record key provides an access path to records in an indexed file.

If the FILE STATUS clause is specified in the file control entry, a value is placed into the specified two-character data item during the execution of an OPEN, CLOSE, READ, WRITE, REWRITE, DELETE or START statement, and before any applicable USE procedure is executed. This is done to indicate to the COBOL program the status of that input-output operation.

The leftmost character position of the FILE STATUS data item is known as status key 1 and is set to indicate one of the following conditions upon completion of the input-output operation.

'0' indicates Successful Completion
'1' indicates At End
'2' indicates Invalid Key
'3' indicates Parity Error
'9' indicates Open Error

The rightmost character position of the FILE STATUS data item is known as status key 2 and is used to further describe the results of the input-output operation. This character will contain one of the following values:

'0' no further information is available concerning the input-output operation.

'2' indicates a duplicate key when status key 1 contains a value of '0'. The duplicate key condition indicates:

a. For a READ statement the key value for the current key of reference is equal to the value of that same key in the next record within the current key of reference.

b. For a WRITE or REWRITE statement, the record just written created a duplicate key value.

'1' indicates a sequence error for a sequentially accessed indexed file when status key 1 contains a value of '2'. Status key 2 will designate the cause of the INVALID KEY condition generated by status key 1.

'3' indicates no record found when status key 1 contains a value of '2'. That is, an attempt has been made to access a record, identified by a key, and that record does not exist in the file.

Data-name-2 must be defined in the Data Division as a two-character data item of the category alphanumeric.

Data-name-1 and data-name-2 can be qualified.

The data item referenced by data-name-1 must be defined as a data item of the category alphanumeric within a record description entry associated with that file-name.

Data-name-1 cannot describe an item whose size is variable.

The File Description Entry for indexed files is the same as sequentially organized files, with the exception that three extra attributes pertaining only to indexed files may have their initial values specified in the "VALUE OF" clause in the file description entry:

```
{VA        }                {ID            }    {data-name-1}
{VALUE     }  OF            {IDENTIFICATION}  IS {literal-1   }
{VALUES    }
```

```
[, KEYSPERENTRY    is   integer-1]
[, AREAOVERFLOW    is   integer-2]
[, FILEOVERFLOW    is   integer-3]
```

Integer-1 represents the number of records that will be written into the file for each fine table entry.

Integer-2 represents the number of overflow records that will be allocated for each area.

Integer-3 represents the number of overflow areas that will be allocated for the file.

For a further discussion, see the ISAM manual.

The following Indexed I-O statements are allowed during the following File Access Modes and Open Mode:

File Access Mode	Statement	PERMISSIBLE STATEMENTS Open Mode Input	Output	Input-Output
Sequential	READ	X		X
	WRITE		X	
	REWRITE			X
	START	X		X
	DELETE			X
Random	READ	X		X
	WRITE		X	X
	REWRITE			X
	START			
	DELETE			X
Dynamic	READ	X		X
	WRITE		X	X
	REWRITE			X
	START	X		X
	DELETE			X

CLOSE

The CLOSE statement, when used as a part of the Indexed I-O module, terminates the processing of indexed files.

The format is as follows:

```
CLOSE  file-name-1 [WITH {LOCK  }]
                         {PURGE}
   [,  file-name-2 [WITH {LOCK  }]  ...
                         {PURGE}
```

The files referenced in the CLOSE statement need not all have the same organization or access.

A CLOSE statement may only be executed for a file in an open mode.

The PURGE option removes the file from the system's directory.

CLOSE and CLOSE WITH LOCK both cause a file to be made a permanent file in the system's directory.

DELETE

The DELETE statement is implemented as part of the ANSI 74 Indexed I/O module. This statement logically removes a record from an indexed file.

The format is as follows:

DELETE file-name RECORD [;INVALID KEY imperative-statement]

The INVALID KEY phrase must not be specified for a DELETE statement which references a file which is in sequential access mode.

The INVALID KEY phrase must be specified for a DELETE statement which references a file which is not in sequential access mode and for which an applicable USE procedure is not specified.

The following are general rules for the DELETE statement:

a. The associated file must be open in the I/O mode at the time of the execution of this statement.

b. For files in the sequential access mode, the last input-output statement executed for file-name prior to the execution of the DELETE statement must have been a successfully executed READ statement.

c. For a file in random or dynamic access mode, the record logically removed from the file is the record identified by the contents of the record key data item associated with file-name. If the file does not contain the record specified by the key, an INVALID KEY condition exists.

After the successful execution of a DELETE statement, the identified record has been logically removed from the file and can no longer be accessed.

The execution of a DELETE statement does not affect the contents of the record area associated with file-name.

The execution of the DELETE statement causes the value of the specified FILE STATUS data item associated with file-name to be updated.

OPEN

The OPEN statement, when used as part of the ANSI 74 Indexed I-O module, initiates the processing of Indexed files.

The general format is as follows:

```
                  ⎧ INPUT  file-name-1  [, file-name-2] ... ⎫
OPEN              ⎨ OUTPUT file-name-3  [, file-name-4] ... ⎬ ...
                  ⎩ I-O    file-name-5  [, file-name-6] ... ⎭
```

The ANSI 74 Indexed I-O statement functions the same as the standard B 7000 / B 6000 OPEN statement.

READ

For sequential access, the READ statement, which is a part of the ANSI 74 Indexed I/O module, makes available the next logical record from a file. For random access, the READ statement makes available a specified record from a mass storage file.

The formats for the READ statement are:

Option 1:

```
READ  file-name [NEXT] RECORD [INTO  identifier]
    [;AT END imperative-statement]
```

Option 2:

```
READ  file-name  RECORD [INTO  identifier]
    [;KEY IS data-name]
    [;INVALID KEY  imperative-statement]
```

Option 1 is the sequential READ statement, and must be used for all files in sequential mode.

In Option 1, the NEXT phrase must be specified for files in dynamic access mode, when records are to be retrieved sequentially.

The INTO phrase must not be used when the input file contains logical records of various sizes as indicated by their record descriptions. The storage area associated with identifier and the storage area which is the record area associated with file-name must not be the same storage area.

Option 2 is the KEYed READ statement, and is used for files in random access mode or for files in dynamic access mode when records are to be retrieved randomly.

Data name must be the name of a data item specified as a record key associated with file-name.

Data-name can be qualified.

The INVALID KEY phrase or the AT END phrase must be specified if no applicable USE procedure is specified for file-name.

The following general rules apply to the INDEXED I-O READ statement:

The record to be made available by an Option 1 READ statement is the next existing record in the file.

The execution of the READ statement causes the value of the FILE STATUS data item associated with file-name to be updated.

When the AT END condition is recognized the following actions are taken in the specified order:

a. A value is placed into the FILE STATUS data item, if specified for this file, to indicate an AT END condition.
b. If the AT END phrase is specified in the statement causing the condition, control is transferred to the AT END imperative statement. Any USE procedure specified for this file is not executed.
c. If the AT END phrase is not specified, then a USE procedure must be specified, either explicitly or implicitly, for this file, and that procedure is executed.

When the AT END condition occurs, execution of the input-output statement which caused the condition is unsuccessful.

Following the unsuccessful execution of any READ statement, the contents of the associated record area are undisturbed.

When the AT END condition has been recognized, an Option 1 READ statement for that file must not be executed without first executing one of the following:

a. A successful CLOSE statement followed by the execution of a successful OPEN statement for that file.
b. A successful START statement for that file.
c. A successful Option 2 READ statement for that file.

For a file in which dynamic access mode is specified, an Option 1 READ statement with the NEXT phrase specified causes the next logically sequential record to be retrieved from that file.

Execution of an Option 2 READ statement causes the value of the key of reference to be compared with the value contained in the corresponding data item of the stored records in the file, until the first record having an equal value is found. This record is then made available. If no record can be so identified, the INVALID KEY condition exists and execution of the READ statement is unsuccessful.

NOTE: The KEY IS data-name clause is for documentation purposes only. The key declared to be the RECORD KEY in the SELECT statement is the only valid key for any I-O on an indexed file.

REWRITE

The REWRITE statement logically replaces a record existing in an Indexed file. This statement is a part of the ANSI 74 specification for the Indexed I/O module. The format is as follows:

```
REWRITE  record-name [FROM  identifier]
    [; INVALID KEY imperative-statement]
```

The record-name and identifier must not refer to the same storage area.

Record-name is the name of a logical record in the File Section of the Data Division and can be qualified.

The INVALID KEY phrase must be specified in the REWRITE statement for files for which an appropriate USE procedure is not specified.

The following general rules apply to the REWRITE statement:

1. The file associated with the record-name must be open in the I-O mode at the time of execution of this statement.
2. For files in the sequential access mode, the last input-output statement executed for the associated file prior to the execution of the REWRITE statement must have been a successfully executed READ statement.
3. The execution of the REWRITE statement causes the value of the FILE STATUS data item associated with the file to be updated.
4. For a file in the sequential access mode, the record to be replaced is specified by the value contained in the record key. When the REWRITE statement is executed, the value contained in the record key data item of the record to be replaced must be equal to the value of the record key of the last record read from the file.
5. For a file in the random or dynamic access mode, the record to be replaced is specified by the record key data item.

   THE INVALID KEY condition exists when:

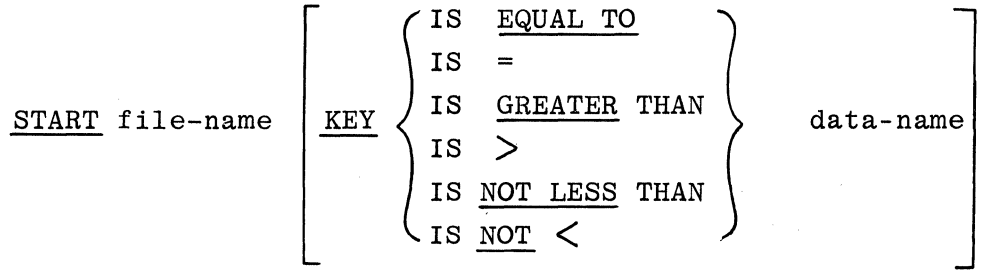
   a. The sequential access mode is specified and the value contained in the record key data item of the record to be replaced is not equal to the value of the record key of the last record read from this file: or

   b. The value contained in the Record Key data item of the Record is not equal to the value of any record in the file.

START

The START statement as part of the ANSI 74 specification of the Indexed I/O module, provides a basis for logical positioning within an indexed file, for subsequent sequential retrieval of records.

The format is as follows:



[; INVALID KEY imperative-statement]

NOTE: The required relational character ' > ', ' < ' and '=' are not underlined to avoid confusion with other symbols.

File-name must be the name of an indexed file.

File-name must be the name of a file with sequential or dynamic access.

Data-name can be qualified.

The INVALID KEY phrase must be specified if no applicable USE procedure is specified for file-name.

If file-name is the name of an indexed file, and if the KEY phrase is specified, data-name may reference a data item specified as a record key associated with file-name, or it may reference any data item of category alphanumeric subordinate to the data-name of a data item specified as a record key associated with file-name whose leftmost character position corresponds to the leftmost character position of the record key data item.

The following general rules apply to the START statement:

1. File-name must be open in the INPUT or I/O mode at the time that the START statement is executed.
2. If the KEY phrase is not specified, the relational operator 'IS EQUAL TO' is implied.

3. The type of comparison specified by the relational operator in the KEY phrase occurs between the key associated with a record in the file referenced by file-name and the specified data-item. If file-name references an indexed file and the operands are of unequal size, comparison, proceeds as though the longer one was truncated on the right such that its length is equal to that of the shorter.

   a. The file is positioned to the first logical record currently existing in the file whose key satisfies the comparison.

   b. If the comparison is not satisfied by any record in the file, an INVALID KEY condition exists, the execution of the START statement is unsuccessful, and no positioning of the file takes place.

The execution of the START statement causes the value of the FILE STATUS data item associated with file-name to be updated.

USE

The USE statement specifies procedures for input-output error handling that are in addition to the standard procedures provided by the input-output control system. This statement is a part of the ANSI 74 specification of the Indexed I/O module.

The format is as follows:

```
USE AFTER STANDARD { EXCEPTION } PROCEDURE ON { file-name-1 [,file-name-2] ... }.
                   { ERROR     }              { INPUT                          }
                                              { OUTPUT                         }
                                              { I-O                            }
```

The words ERROR and EXCEPTION are synonymous and may be used interchangeably.

The files implicitly or explicitly referenced in a USE statement need not all have the same organization or access.

The following general rules apply to the USE statement.

The designated procedures are executed by the input-output system after completing the standard input-output error routine, or upon recognition of the INVALID KEY or AT END conditions, when the INVALID KEY phrase or AT END phrase, respectively, has not been specified in the input-output statement.

After execution of a USE procedure, control is returned to the invoking routine.

WRITE

The WRITE statement when used as part of the ANSI 74 specification of the Indexed I/O module, releases a logical record for an output or input-output file.

The format is as follows:

WRITE record-name [FROM identifier] [;INVALID KEY imperative-statement]

Record-name and identifier must not reference the same storage area.

The record-name is the name of a logical record in the File Section of the Data Division and can be qualified.

The INVALID KEY phrase must be specified if an applicable USE procedure is not specified for the associated file.

The following general rules apply to the WRITE statement.

The associated file must be open in the OUTPUT or I/O mode at the time of the execution of this statement.

The execution of the WRITE statement causes the value of the FILE STATUS data item associated with the file to be updated.

If sequential access is specified for the file, records must be released in ascending order of record key values.

If random or dynamic access mode is specified, records may be released in any program-specified order.

When a file is opened in the output mode, records must be written with ascending keys, regardless of the access mode.

The INVALID KEY condition exists under the following circumstances.

a. When a file is opened in the output mode, regardless of the access mode, and the value of the record key is not greater than the value of the record key of the previous record, or
b. When the file is opened in the output or I/O mode, and the value of the record key is equal to the value of a record key of a record already existing in the file.

When the INVALID KEY condition is recognized, the execution of the WRITE statement is unsuccessful. Also, the contents of the record area are unaffected and the FILE STATUS data item associated with the file-name of the associated file is set to a value indicating the cause of the condition. Execution of the program proceeds according to the rules stated in the INVALID KEY condition.

SEQUENTIAL I-O

The following ANSI 74 Sequential I-O features have been added to existing B 7000/B 6000 COBOL syntax and semantics:

The CODE-SET clause in the FILE DESCRIPTION ENTRIES.

The EXTEND option of the CLOSE statement.

The ORGANIZATION/FILE STATUS clauses of the FILE-CONTROL paragraph.

LINAGE

The LINAGE clause, when used as part of the ANSI 74 Sequential I-O Module, can be used only when the ANSI74 system dollar option is set.

The LINAGE clause provides a means for specifying the depth of a logical page in terms of numbers of lines. It also provides for specifying the size of the top and bottom margins on the logical page, and the line number, within the page body, at which the footing area begins.

The general format is as follows:

```
LINAGE IS {data-name-1 } LINES [, WITH FOOTING AT {data-name-2}]
          {integer-1   }       [                  {integer-2  }]
[, LINES AT TOP {data-name-3}][, LINES AT BOTTOM {data-name-4}]
               {integer-3  }                     {integer-4  }
```

Data-name-1, data-name-2, data-name-3, data-name-4 must reference elementary unsigned numeric integer data items.

The value of integer-1 must be greater than zero.

The value of integer-2 must not be greater than integer-1.

The value of integer-3, integer-4 can be zero.

If the ANSI74 system dollar option is set at the time the LINAGE clause is used in the File Description, the semantic meaning of the ANSI 74 LINAGE clause will be given to the file.

If the ANSI74 system dollar option is not set, the current semantic meaning of the LINAGE clause will be given to the file.

The following general rules apply to the use of the ANSI 74 extension of the LINAGE clause.

1. The LINAGE clause provides a means for specifying the size of a logical page in terms of number of lines. The logical page size is the sum of the values referenced by each phrase except the FOOTING phrase. If the LINES AT TOP or LINES AT BOTTOM phrases are not specified, the values for these functions are zero. If the FOOTING phrase is not specified, the assumed value is equal to integer-1, or the contents of the data item referenced by data-name-1, whichever is specified.

There is not necessarily any relationship between the size of the logical page and the size of a physical page.

2. The value of integer-1 or the data item referenced by data-name-1 specifies the number of lines that can be written and/or spaced on the logical page. The value must be greater than zero. That part of the logical page in which these lines can be written and/or spaced is called the page body.
3. The value of integer-3 or the data item referenced by data-name-3 specifies the number of lines that comprise the top margin on the logical page. The value may be zero.
4. The value of integer-4 or the data item referenced by data-name-4 specifies the number of lines that comprise the bottom margin on the logical page. The value may be zero.
5. The value of integer-2 or the data item referenced by data-name-2 specifies the line number within the page body at which the footing area begins. The value must be greater than zero and not greater than the value of integer-1 or the data item referenced by data-name-1.

   The footing area comprises the area of the logical page between the line represented by the value of integer-2 or the data item referenced by data-name-2 and the line represented by the value of integer-1 or the data item referenced by data-name-1, inclusive.
6. The value of integer-1, integer-3, and integer-4 if specified, will be used at the time the file is opened by the execution of an OPEN statement with the OUTPUT phrase, to specify the number of lines that comprise each of the indicated sections of a logical page. The value of integer-2, if specified, will be used at that time to define the footing area. These values are used for all logical pages written for the file during a given execution of the program.
7. The values of the data items referenced by data-name-1, data-name-3, and data-name-4, if specified, will be used as follows:

   a. The values of the data items, at the time an OPEN statement with the OUTPUT phrase is executed for the file, will be used to specify the number of lines that are to comprise each of the indicated sections for the first logical page.
   b. The values of the data items, at the time a WRITE statement with the ADVANCING PAGE phrase is executed or page overflow condition occurs, will be used to specify the number of lines that are to comprise each of the indicated sections for the next logical page.

8. The value of the data item referenced by data-name-2, if specified, at the time an OPEN statement with the OUTPUT phrase is executed for the file, will be used to define the footing area for the first logical page. At the time a WRITE statement with the ADVANCING PAGE phrase is executed or a page overflow condition occurs, it will be used to define the footing area for the next logical page.
9. A LINAGE-COUNTER is generated by the presence of a LINAGE clause. The value in the LINAGE-COUNTER at any given time represents the line number at which the device is positioned within the current page body. The rules governing the LINAGE-COUNTER are as follows:
   a. A separate LINAGE-COUNTER is supplied for each file described in the File Section whose file description entry contains a LINAGE clause.
   b. LINAGE-COUNTER may be referenced, but may not be modified, by Procedure Division statements. Since more than one LINAGE-COUNTER may exist in a program, the user must qualify LINAGE-COUNTER by file-name when necessary.
   c. LINAGE-COUNTER is automatically modified, according to the following rules, during the execution of a WRITE statement to an associated file:
      1) When the ADVANCING PAGE phrase of the WRITE statement is specified, the LINAGE-COUNTER is automatically reset to one (1).
      2) When the ADVANCING identiier-2 or integer phrase of the WRITE statement is specified, the LINAGE-COUNTER is incremented by integer or the value of the data item referenced by identifier-2.
      3) When the ADVANCING phrase of the WRITE statement is not specified, the LINAGE-COUNTER is incremented by the value one (1).
      4) The value of LINAGE-COUNTER is automatically reset to one (1) when the device is repositioned to the first line that can be written on for each of the succeeding logical pages.
   d. The value of LINAGE-COUNTER is automatically set to one (1) at the time an OPEN statement is executed for the associated file.
10. Each logical page is contiguous to the next with no additional spacing provided. (This assumes that no automatic channel advance mechanism is present on a hardware device).

WRITE

The following semantic actions for the WRITE statement are related to the specification of the ANSI 74 LINAGE clause.

The syntax for the WRITE statement associated with the LINAGE clause for Sequential I-O is as follows:

```
WRITE  record-name [FROM identifier-1]

[ {BEFORE}                {identifier-2}      [LINE ]   ]
[ {AFTER } ADVANCING      {integer     }      [LINES]   ]
[                         {mnemonic-name}                ]
[                         {PAGE        }                ]

[ ; AT {END-OF-PAGE} imperative-statement ]
       {EOP        }
```

If the END-OF-PAGE phrase is specified, the LINAGE clause must be specified in the file description entry for the file.

The "ADVANCING mnemonic-name" phrase cannot be specified for a WRITE statement referencing a file whose description contains a LINAGE clause.

Both the ADVANCING phrase and the END-OF-PAGE phrase allow control of the vertical positioning of each line on a representation of a printed page. If the ADVANCING phrase is not used, automatic advancing will be provided by the implementor to act as if the user had specified AFTER ADVANCING 1 LINE. If the ADVANCING phrase is used, advancing is provided as follows:

a. If "ADVANCING identifier-2 LINES" is specified, the representation of the printed page is advanced the number of lines equal to the current value associated with identifier-2.
b. If "ADVANCING integer LINES" is specified, the representation of the printed page is advanced the number of lines equal to the value of integer.
c. If the BEFORE phrase is used, the line is presented before the representation of the printed page is advanced according to rules a and b.
d. If the AFTER phrase is used, the line is presented after the representation of the printed page is advanced according to rules a and b.

e. If the "ADVANCING PAGE" phrase is specified, the record is presented on the logical page before or after (depending on the phrase used) the device is repositioned to the next logical page. If the record to be written is associated with a file whose file description entry contains a LINAGE clause, the repositioning is to the first line that can be written on the next logical page as specified in the LINAGE clause.

If the logical end of the representation of the printed page is reached during the execution of a WRITE statement with the END-OF-PAGE phrase, the imperative-statement specified in the END-OF-PAGE phrase is executed. The logical end is specified in the LINAGE clause associated with record-name.

An end-of-page condition is reached whenever the execution of a given WRITE statement with the END-OF-PAGE phrase causes printing or spacing within the footing area of a page body. This occurs when the execution of such a WRITE statement causes the LINAGE-COUNTER to equal or exceed the value specified by integer-2 or the data item referenced by data-name-2 of the LINAGE clause, if specified. In this case, the WRITE statement is executed and the imperative statement in the END-OF-PAGE phrase is executed.

An automatic page overflow condition is reached whenever the execution of a given WRITE statement (with or without an END-OF-PAGE phrase) cannot be fully accommodated within the current page body.

This occurs when a WRITE statement, if executed, would cause the LINAGE-COUNTER to exceed the value specified by integer-1 or the data item referenced by data-name-1 of the LINAGE clause. In this case, the record is presented on the logical page before or after (depending on the phrase used) the device is repositioned to the first line that can be written on the next logical page as specified in the LINAGE clause. The imperative statement in the END-OF-PAGE clause, if specified, is executed after the record is written and the device has been repositioned.

If integer-2 or data-name-2 of the LINAGE clause is not specified, no end-of-page condition distinct from the page overflow condition is detected. In this case, the end-of-page condition and page overflow condition occur simultaneously.

If integer-2 or data-name-2 of the LINAGE clause is specified, but the execution of a given WRITE statement would cause LINAGE-COUNTER to simultaneously exceed the value of both integer-2 or the data item referenced by data-name-2 and integer-1 or the data item referenced by data-name-1, then the operation proceeds as if integer-2 or data-name-2 had not been specified.

TABLE HANDLING

The following Table Handling feature has been added to existing B 7000 / B 6000 COBOL syntax and semantics:

The DEPENDING option extension in the OCCURS clause.

There are no other differences between existing language elements and ANSI 74 language elements with regards to the Table Handling Module.

NUCLEUS

The following ANSI 74 Nucleus features have been added to existing B 7000 / B 6000 COBOL syntax and semantics:

Option 2 of the ACCEPT statement in the PROCEDURE DIVISION.

The alphabet-name clause in the SPECIAL-NAMES paragraph.

The maximum decimal value restrictions for COMPUTATIONAL data items.

The meaning of the NOT operator in abbreviated relation conditions.

The PROGRAM COLLATING SEQUENCE in the OBJECT-COMPUTER paragraph.

The "/" insertion character in the SIMPLE INSERTION EDITING sub-section in the PICTURE section.

The following ANSI 74 Nucleus features are additions to the above mentioned ANSI 74 features and are described in detail on the following pages.

INSPECT

The INSPECT statement provides the ability to tally (Option 1), replace (Option 2) or tally and replace (Option 3) occurrences of single characters or groups of characters in a data item.

The general format is as follows:

Option 1:

```
INSPECT identifier-1 TALLYING
    [ , identifier-2 FOR [ , [ ALL       ] [ identifier-3 ] ] [ [ BEFORE ] INITIAL [ identifier-4 ] ] ... ] ...
                         [ LEADING   ] [ literal-1    ]   [ [ AFTER  ]         [ literal-2    ] ]
                         [ CHARACTERS ]
```

Option 2:

```
INSPECT identifier-1 REPLACING
    { CHARACTERS BY [ identifier-6 ] [ BEFORE ] INITIAL [ identifier-7 ]                                                                                    }
                   [ literal-4    ] [ AFTER  ]         [ literal-5    ]                                                                                    }
    { ( , [ ALL       ] ) [ , identifier-5 ] BY [ identifier-6 ] [ [ BEFORE ] INITIAL [ identifier-7 ] ] ... } ...
      ( , [ LEADING   ] )   [ literal-3    ]    [ literal-4    ]   [ [ AFTER  ]         [ literal-5    ] ]
      ( , [ FIRST     ] )
```

Option 3:

```
INSPECT identifier-1 TALLYING
    [ , identifier-2 FOR [ , [ ALL       ] [ identifier-3 ] ] [ [ BEFORE ] INITIAL [ identifier-4 ] ] ... ] ...
                         [ LEADING   ] [ literal-1    ]   [ [ AFTER  ]         [ literal-2    ] ]
                         [ CHARACTERS ]
    REPLACING
    { CHARACTERS BY [ identifier-6 ] [ BEFORE ] INITIAL [ identifier-7 ]                                                                                    }
                   [ literal-4    ] [ AFTER  ]         [ literal-5    ]                                                                                    }
    { ( , [ ALL       ] ) [ , identifier-5 ] BY [ identifier-6 ] [ [ BEFORE ] INITIAL [ identifier-7 ] ] ... } ...
      ( , [ LEADING   ] )   [ literal-3    ]    [ literal-4    ]   [ [ AFTER  ]         [ literal-5    ] ]
      ( , [ FIRST     ] )
```

All Options

Identifier-1 must reference either a group item or any category of elementary item, described as usage DISPLAY.

Identifier-3 . . . identifier-n must reference either an elementary alphabetic, alphanumeric or numeric item described as usage DISPLAY

Each literal must be nonnumeric and may be any figurative constant, except ALL.

Literal-1, literal-2, literal-3, literal-4 and literal-5, and the data items referenced by identifier-3, identifier-4, identifier-5, identifier-6 and identifier-7 may be one or more characters in length, except as specifically noted in syntax and general rules.

Options 1 and 3 only:

Identifier-2 must reference on elementary numeric data item.

If either literal-1 or literal-2 is a figurative constant, the figurative constant refers to an implicit one character data item.

Options 2 and 3 only:

The size of the data referenced by literal-4 or identifier-6 must be equal to the size of the data referenced by literal-3 or identifier-5. When a figurative constant is used as literal-4, the size of the figurative constant is equal to the size of literal-3 or the size of the data item referenced by identifier-5.

When the CHARACTERS phrase is used, literal-4 literal-5 or the size of the data item referenced by identifier-6, identifier-7 must be one character in length.

When a figurative constant is used as literal-3, the data referenced by literal-4 or identifier-6 must be one character in length.

The following general rules apply to the INSPECT statement.

1. Inspection (which includes the comparison cycle, the establishment of boundaries for the BEFORE or AFTER phrase, and the mechanism for tallying and/or replacing) begins at the leftmost character position of the data item referenced by identifier-1, regardless of its class, and proceeds from left to right to the rightmost character position as described in general rules 4 through 6.

2. For use in the INSPECT statement, the contents of identifier-1, identifier-3, identifier-4, identifier-5, identifier-6 or identifier-7 will be treated as follows:

   a. If any of identifier-1, identifier-3, identifier-4, identifier-5, identifier-6 or identifier-7 are described as alphanumeric, the INSPECT statement treats the contents of each such identifier as a character-string.
   b. If any of identifier-1, identifier-3, identifier-4, identifier-5, identifier-6 or identifier-7 are described as alphanumeric edited, numeric edited or unsigned numeric, the data item is inspected as though it had been redevined as alphanumeric and the INSPECT statement had been written to reference the redefined data item.
   c. If any of the identifier-1, identifier-3, identifier-4, identifier-5, identifier-6 or identifier-7 are described as signed numeric, the data item is inspected as though it has been moved to an unsigned numeric data item of the same length and then general rule 2-b had been applied.

3. In general rules 4 through 11 all references to literal-1, literal-2, literal-3, literal-4 and literal-5 apply equally to the contents of the data item referenced by identifier-3, identifier-4, identifier-5, identifier-6 and identifier-7, respectively.
4. During inspection of the contents of the data item referenced by identifier-1, each properly matched occurrence of literal-1 is tallied (Options 1 and 3) and/or each properly matched occurrence of literal-3 is replaced by literal-4 (Options 2 and 3).
5. The comparison operation to determine the occurrences of literal-1 to be tallied, and/or occurrences of literal-3 to be replaced, occurs as follows:

   a. The operands of the TALLYING and REPLACING phrases are considered in the order they are specified in the INSPECT statement from left to right. The first literal-1, literal-3, is compared to an equal number of contiguous characters, starting with the leftmost character position in the data item referenced by identifier-1, literal-1, literal-3 and that portion of the contents of the data item referenced by identifier-1 match if, and only if, they are equal, character for character.

b. If no match occurs in the comparison of the first literal-1, literal-3, the comparison is repeated with each successive literal-1, literal-3, if any, until either a match is found or there is no next successive literal-1, literal-3. When there is no next successive literal-1, literal-3, the character position in the data item referenced by identifier-1 immediately to the right of the leftmost character position considered in the last comparison cycle is considered as the leftmost character position, and the comparison cycle begins again with the first literal-1, literal-3.

c. Whenever a match occurs, tallying and/or replacing takes place as described in general rules 8 through 10. The character position in the data item referenced by identifier-1 immediately to the right of the rightmost character position that participated in the match is now considered to be the leftmost character position of the data item referenced by identifier-1, and the comparison cycle starts again with the first literal-1, literal-3.

d. The comparison operation continues until the rightmost character position of the data item referenced by identifier-1 has participated in a match or has been considered as the leftmost character position. When this occurs, inspection is terminated.

e. If the CHARACTERS phrase is specified, an implied one-character operand participates in the cycle described in paragraphs 5-a through 5-d above, except that no comparison to the contents of the data item referenced by identifier-1 takes place. This implied character is considered always to match the leftmost character of the contents of the data item referenced by identifier-1 participating in the current comparison cycle.

6. The comparison operation defined in general rule 5 is affected by the BEFORE and AFTER phrases as follows:

a. If the BEFORE or AFTER phrase is not specified, literal-1, literal-3 or the implied operand of the CHARACTERS phrase participates in the comparison operations as described in general rule 5.

b. If the BEFORE phrase is specified, the associated literal-1, literal-3 or the implied operand of the CHARACTERS phrase participates only in those comparison cycles which involve that portion of the contents of the data item referenced by identifier-1 from its leftmost character position up to, but not

including, the first occurrence of literal-2, literal-5 within the contents of the data item referenced by identifier-1. The position of this first occurrence is determined before the first cycle of the comparison operation described in general rule 5 is begun. If, on any comparison cycle, literal-1, literal-3 or the implied operand of the CHARACTERS phrase is not eligible to participate, it is considered not to match the contents of the data item referenced by identifier-1. If there is no occurrence of literal-2, literal-5 within the contents of the data item referenced by identifier-1, its associated literal-1, literal-3, or the implied operand of the CHARACTERS phrase participates in the comparison operation as though the BEFORE phrase had not been specified.

c. If the AFTER phrase is specified, the associated literal-1, literal-3 or the implied operand of the CHARACTERS phrase may participate only in those comparison cycles which involve that portion of the contents of the data item referenced by identifier-1 from the character position immediately to the right of the rightmost character position of the first occurrence of literal-2, literal-5 within the contents of the data item referenced by identifier-1 and the rightmost character position of the data item referenced by identifier-1. The position of this first occurrence is determined before the first cycle of the comparison operation described in general rule 5 is begun. If, on any comparison cycle, literal-1, literal-3 or the implied operand of the CHARACTERS phrase is not eligible to participate, it is considered not to match the contents of the data item referenced by identifier-1. If there is no occurrence of literal-2, literal-5 within the contents of the data item referenced by identifier-1, its associated literal-1, literal-3, or the implied operand of the CHARACTERS phrase is never eligible to participate in the comparison operation.

Option 1:

7. The contents of the data item referenced by identifier-2 is not initialized by the execution of the INSPECT statement.
8. The rules for tallying are as follows:

   a. If the ALL phrase is specified, the contents of the data item referenced by identifier-2 is incremented by one (1) for each

occurrence of literal-1 matched within the contents of the data item referenced by identifier-1.

b. If the LEADING phrase is specified, the contents of the data item referenced by identifier-2 is incremented by one (1) for each contiguous occurrence of literal-1 matched within the contents of the data item referenced by identifier-1, provided that the leftmost such occurrence is at the point where comparison began in the first comparison cycle in which literal-1 was eligible to participate.

c. If the CHARACTERS phrase is specified, the contents of the data item referenced by identifier-2 is incremented by one (1) for each character matched, in the sense of general rule 5-e, within the contents of the data item referenced by identifier-1.

Option 2:

9. The required words ALL, LEADING and FIRST are adjectives that apply to each succeeding BY phrase until the next adjective appears.

10. The rules for replacement are as follows:

a. When the CHARACTERS phrase is specified, each character matched, in the sense of general rule 5-e, in the contents of the data item referenced by identifier-1 is replaced by literal-4.

b. When the adjective ALL is specified, each occurrence of literal-3 matched in the contents of the data item referenced by identifier-1 is replaced by literal-4.

c. When the adjective LEADING is specified, each contiguous occurrence of literal-3 matched in the contents of the data item referenced by identifier-1 is replaced by literal-4, provided that the leftmost occurrence is at the point where comparison began in the first comparison cycle in which literal-3 was eligible to participate.

d. When the adjective FIRST is specified, the leftmost occurrence of literal-3 matched within the contents of the data item referenced by identifier-1 is replaced by literal-4.

Option 3:

11. An Option 3 INSPECT statement is interpreted and executed as though two successive INSPECT statements specifying the same identifier-1 had been written with one statement being an Option 1 statement with TALLYING phrases identical to those specified in the Option 3

statement, and the other statement being an Option 2 statement with REPLACING phrases identical to those specified in the Option 3 statement. The general rules given for matching and counting apply to the Option 1 statement and the general rules given for matching and replacing apply to the Option 2 statement.

SIGN

The SIGN clause is a ANSI 74 Nucleus module facility which specifies the position and mode of representation of the operational sign when it is necessary to describe these properties explicitly. The SIGN clause is used in conjunction with the PICTURE clause in a data description.

The format is as follows:

$$[\underline{\text{SIGN}}\ \text{IS}] \left\{ \begin{matrix} \underline{\text{LEADING}} \\ \underline{\text{TRAILING}} \end{matrix} \right\} [\underline{\text{SEPARATE}}\ \text{CHARACTER}]$$

The SIGN clause may be specified only for a numeric data-item whose PICTURE contains the character 's', or a group item with such entries subordinate.

The data description entries to which the SIGN clause applies must be of display usage.

At most one SIGN clause may apply to any given numeric data description entry.

The specification of "TRAILING SEPARATE CHARACTER" is not implemented.

If the CODE-SET clause is specified, any signed numeric data description entries associated with that file description entry must be described with the SIGN IS SEPARATE clause.

The following general rules apply to the use of the SIGN clause.

(1) The optional SIGN clause, if present, specifies the position and the mode of representation of the operational sign for the numeric data description entry to which it applies, or for each numeric data description entry subordinate to the group to which it applies. The SIGN clause applies only to numeric data description entries whose PICTURE contains the character 's'.

(2) A numeric DISPLAY data description entry whose PICTURE contains the character 's', but to which no optional SIGN clause applies, has an operational sign. In this (default) case, the sign is in the zone of the trailing character.

(3) If the operational SEPARATE CHARACTER phrase is not present, then:

    a. The operational sign will be presumed to be the zone of the leading (or, respectively, trailing) character position of the elementary numeric data item.

b. The letter 's' in a PICTURE character-string is not counted in determining the size of the item (in terms of standard data format characters).

(4) If the optional SEPARATE CHARACTER phrase is present, then:

a. The operational sign will be presumed to be the leading character position of the elementary numeric data item; this character position is not a digit position.

b. The letter 's' in a PICTURE character-string is counted in determining the size of the item (in terms of standard data format characters).

c. The operational signs for positive and negative are the standard DISPLAY characters '+' and '-', respectively.

(5) Every numeric data description entry whose PICTURE contains the character 's' is a signed numeric data description entry. If a SIGN clause applies to such an entry and conversion is necessary for purposes of computation or comparisons, conversion takes place automatically.

STRING

The STRING statement provides juxtaposition of the partial or complete contents of two or more data items into a single data item.

The general format is as follows:

```
STRING { identifier-1 } [, { identifier-2 } ]...  DELIMITED BY { identifier-3 }
       { literal-1    }     { literal-2    }                    { literal-3    }
                                                               { SIZE         }

    [, { identifier-4 } [, { identifier-5 } ]...  DELIMITED BY { identifier-6 } ]...
       { literal-4    }     { literal-5    }                    { literal-6    }
                                                               { SIZE         }

  INTO identifier-7 [WITH POINTER identifier-8]

  [; ON OVERFLOW imperative-statement]
```

Each literal may be any figurative constant without the optional word ALL.

All literals must be described as nonnumeric literals; also, all identifiers, except identifier-8, must be described implicitly or explicitly as usage DISPLAY.

Identifier-7 must represent an alphanumeric data item without editing symbols or the JUSTIFIED clause.

Identifier-8 must represent an elementary numeric integer data item of sufficient size to contain a value equal to the size plus 1 of the area referenced by identifier-7. The symbol 'P' may not be used in the PICTURE character-string of identifier-8.

Where identifier-1, identifier-2, ..., or identifier-3 is an elementary numeric data item, it must be described as an integer without the symbol 'P' in its PICTURE character-string.

The general rules for the STRING statement are as follows:

1. All references to identifier-1, identifier-2, identifier-3, literal-1, literal-2, literal-3 apply equally to identifier-4, identifier-5, identifier-6, literal-4, literal-5 and literal-6, respectively, and all iterations thereof.
2. Identifier-1, literal-1, identifier-2, literal-2 represent the sending items. Identifier-7 represents the receiving item.
3. Literal-3, identifier-3, indicate the character(s) delimiting the move. If the SIZE phrase is used, the complete data item defined by identifier-1, literal-1, identifier-2, literal-2, is moved.

When a figurative constant is used as the delimiter, it stands for a single-character nonnumeric literal.

4. When a figurative constant is specified as literal-1, literal-2, literal-3, it refers to an implicit one-character data item whose usage is DISPLAY.
5. When the STRING statement is executed, the transfer of data is governed by the following rules:
   a. Those characters from literal-1, literal-2, or from the contents of the data item referenced by identifier-1, identifier-2, are transferred to the contents of identifier-7 in accordance with the rules for alphanumeric to alphanumeric moves, except that no space-filling will be provided.
   b. If the DELIMITED phrase is specified without the SIZE phrase, the contents of the data item referenced by identifier-1, identifier-2, or the value of literal-1, literal-2, are transferred to the receiving data item in the sequence specified in the STRING statement; thus, beginning with the leftmost character and continuing from left to right until the end of the data item is reached, or until the character(s) specified by literal-3, or by the contents of identifier-3 are encountered. The character(s) specified by literal-3 or by the data item referenced by identifier-3 are not transferred.
   c. If the DELIMITED phrase is specified with the SIZE phrase, the entire contents of literal-1, literal-2, or the contents of the data item referenced by identifier-1, identifier-2 are transferred, in the sequence specified in the STRING statement, to the data item referenced by identifier-7 untl all data has been transferred or the end of the data item referenced by identifier-7 has been reached.
6. If the POINTER phrase is specified, identifier-8 is explicitly available to the programmer, and he is responsible for setting its initial value. The initial value must not be less than one.
7. If the POINTER phrase is not specified, the following general rules apply as if the user had specified identifier-8 with an initial value of 1.

8. Characters transferred to the data item referenced by identifier-7 are moved (seemingly one at a time) from the source into the data item referenced by identifier-7. These characters are designated by the value associated with identifier-8, which is obtained by incrementing the value of identifier-8 by one prior to the move of the next character.
9. At the termination of STRING statement execution, only the portion of identifier-7 referenced during execution is changed. All other portions of identifier-7 will contain data that was present before the execution of the STRING statement.
10. If at the beginning of the string statement execution, the value of identifier-8 is either less than one or greater than the number of character positions in identifier-7, then no data will be transferred, and the imperative statement in the ON OVERFLOW phrase, if specified, will be executed.
11. If the ON OVERFLOW phrase is not specified when the conditions described in general rule 10 are encountered, control is transferred to the next executable statement.

UNSTRING

The UNSTRING statement causes contiguous data in a sending field to be separated and placed into multiple receiving fields.

The general format for the UNSTRING statement is as follows:

```
UNSTRING identifier-1
[DELIMITED BY [ALL] {identifier-2 } [, OR [ALL] {identifier-3 }] ...]
                    {literal-1    }            {literal-2    }
  INTO identifier-4 [,DELIMITER IN identifier-5][,COUNT IN identifier-6]
     [, identifier-7 [,DELIMITER IN identifier-8][,COUNT IN identifier-9] ]...
 [WITH POINTER identifier-10][TALLYING IN identifier-11]
 [; ON OVERFLOW imperative-statement]
```

Each literal must be a nonnumeric literal. In addition, each literal may be any figurative constant without the optional word ALL.

Identifier-1, identifier-2, identifier-3, identifier-5 and identifier-8 must be described implicitly or explicitly, as an alphanumeric DISPLAY data item.

Identifier-4 and identifier-7 may not be described as requiring editing, and must be described as usage DISPLAY.

Identifier-6, identifier-9, identifier-10 and identifier-11 must be described as elementary numeric integer data items (except that the symbol 'P' may not be used in the PICTURE character-string).

No identifier may name a level 88 entry.

The DELIMITER IN phrase and the COUNT IN phrase may be specified only if the DELIMITED BY phrase is specified.

The following general rules apply to the UNSTRING statement:

1. All references to identifier-2, literal-1, identifier-4, identifier-5 and identifier-6, apply equally to identifier-3, literal-2, identifier-7, identifier-8 and identifier-9, respectively, and all iterations thereof.
2. Identifier-1 represents the sending area.
3. Identifier-4 represents the data receiving area. Identifier-5 represents the receiving area for delimiters.

4. Literal-1 or the data item referenced by identifier-2 specifies a delimiter.
5. Identifier-6 represents the count of the number of characters within the data item referenced by identifier-1 isolated by the delimiters for the move to identifier-4. This value does not include a count of the delimiter character(s).
6. The data item referenced by identifier-10 contains a value that indicates a relative character position within the area defined by identifier-1.
7. The data item referenced by identifier-11 is a counter that records the number of data items acted upon during the execution of an UNSTRING statement.
8. When a figurative constant is used as the delimiter, it stands for a single-character nonnumeric literal.

   When the ALL phrase is specified, one occurrence or two more contiguous occurrences of literal-1 (figurative constant or not) or the contents of the data item referenced by identifier-2 are treated as if there were only one occurrence, and is moved to the receiving data item according to general rule 13-d.
9. When any examination encounters two contiguous delimiters, the current receiving area is either space or zero filled according to the description of the receiving area.
10. Literal-1 or the contents of the data item referenced by identifier-2 can contain any character in the EBCDIC character set.
11. Each literal-1 or the data item referenced by identifier-2 represents one delimiter. When a delimiter contains two or more characters, all of the characters must be present in contiguous positions of the sending item, and in the order given to be recognized as a delimiter.
12. When two or more delimiters are specified in the DELIMITED BY phrase, an 'OR' condition exists between them; that is, each delimiter is compared to the sending field. If a match occurs, the character(s) in the sending field is/are considered to be a single delimiter. No character(s) in the sending field can be considered as a part of more than one delimiter.

    Each delimiter is applied to the sending field in the sequence specified in the UNSTRING statement.

Specifically, the strings unstrung will be the same regardless of the order in which the multiple delimiters are specified. For example, given the source string "ABC,DEF GHI" and the delimiting phrase **DELIMITED BY SPACES OR ","**, the strings unstrung from the source string are "ABC", "DEF" and "GHI". This is because all specified delimiters are checked against the first character in the source string before proceeding to the second character of the source string.

13. When the UNSTRING statement is initiated, the current receiving area is the data item referenced by identifier-4. Data is transferred from the data item referenced by identifier-1 to the data item referenced by identifier-4 according to the following rules:

    a. If the POINTER phrase is specified, the string of characters referenced by identifier-1 is examined beginning with the relative character position indicated by the contents of the data item referenced by identifier-10. If the POINTER phrase is not specified, the string of characters is examined beginning with the leftmost character position.

    b. If the DELIMITED BY phrase is specified, the examination proceeds left to right until either a delimiter specified by the value of literal-1 or the data item referenced by identifier-2 is encountered. If the DELIMITED BY phrase is not specified, the number of characters examined is equal to the size of the current receiving area. However, if the sign of the receiving item is defined as occupying a separate character position, the number of characters examined is one less than the size of the current receiving area.

       If the end of the data item referenced by identifier-1 is encountered before the delimiting condition is met, the examination terminates with the last character examined.

    c. The characters thus examined (excluding the delimiting character(s), if any) are treated as an elementary alphanumeric data item, and are moved into the current receiving area according to the rules for the MOVE statement.

    d. If the DELIMITER IN phrase is specified, the delimiting character(s) are treated as an elementary alphanumeric data item and are moved into the data item referenced by identifier-5 according to the rules for the MOVE statement. If

the delimiting condition is the end of the data item referenced by identifier-1, then the data item referenced by identifier-5 is SPACE FILLED.

e. If the COUNT IN phrase is specified, a value equal to the number of characters thus examined (excluding the delimiter character(s), if any) is moved into the area referenced by identifier-6 according to the rules for an elementary move.

f. If the DELIMITED BY phrase is specified, the string of characters is further examined beginning with the first character to the right of the delimiter. If the DELIMITED BY phrase is not specified, the string of characters is further examined beginning with the character to the right of the last character transferred.

g. After data is transferred to the data item referenced by identifier-4, the current receiving area is the data item referenced by identifier-7. The behavior described in paragraph 13-b through 13-f is repeated until either all the characters are exhausted in the data item referenced by identifier-1, or until there are no more receiving areas.

14. The initialization of the contents of the data items associated with the POINTER phrase or the TALLYING phrase is the responsibility of the user.

15. The contents of the data item referenced by identifier-10 will be incremented by one for each character examined in the data item referenced by identifier-1. When the execution of an UNSTRING statement with a POINTER phrase is completed, the contents of the data item referenced by identifier-10 will contain a value equal to the initial value plus the number of characters examined in the data item referenced by identifier-1.

16. When the execution of an UNSTRING statement with a TALLYING phrase is completed, the contents of the data item referenced by identifier-11 contains a value equal to its initial value plus the number of data receiving items acted upon.

17. Either of the following situations causes an overflow condition:

    a. An UNSTRING is initiated, and the value in the data item referenced by identifier-10 is less than 1 or greater than the size of the data item referenced by identifier-1.

b. If, during execution of an UNSTRING statement, all data receiving areas have been acted upon, and the data item referenced by identifier-1 contains characters that have not been examined.

18. When an overflow condition exists, the UNSTRING operation is terminated. If an ON OVERFLOW phrase has been specified, the imperative statement included in the ON OVERFLOW phrase is executed. If the ON OVERFLOWphrase is not specified, control is transferred to the next executable statement.
19. The evaluation of subscripting and indexing for the identifiers is as follows:

    a. Any subscripting or indexing associated with identifier-1, identifier-10, identifier-11 is evaluated only once, immediately before any data is transferred as a result of the execution of the UNSTRING statement.
    b. Any subscripting or indexing associated with identifier-2, identifier-3, identifier-4, identifier-5, identifier-6 is evaluated immediately before the transfer of data into the respective data item.

C. B2500 IMPLEMENTATIONS

The following features have been implemented in B 7000 / B 6000 COBOL to allow more compatibility with B 3700 COBOL. These features are available only while the B2500 system dollar option is set. The default initial value of the B2500 system dollar option may be set to true by compiling the COBOL compiler with "B2500" set.

The KEY CONVERSION clause of the SEEK statement in the PROCEDURE DIVISION section.

The O-I and LOCK ACCESS options of the OPEN statement in the PROCEDURE DIVISION section.

The REMAINDER ROUNDED and MOD options of the DIVIDE statement in the PROCEDURE DIVISION section.

The SAVE option of the SELECT clause in the FILE-CONTROL section.

The PURGE, RUN, END, LOCK, and RELEASE options in the SORT statement in the PROCEDURE DIVISION section.

The letter K in a picture character string in the PICTURE clause.

COMPUTATIONAL to mean 4-bit character data when USAGE IS COMPUTATIONAL is specified.

UNDIGIT literals in the DEFINITION OF WORDS sub-section.

The use of initializing VALUE clauses in the File Section.

Unsigned 4-bit integer items allowed to be declared greater than 23 digits in the DATA DIVISION.

The redefinition of a DISPLAY data item by a COMP-2 data item of different size in the REDEFINES clause in the DATA DIVISION section.

The ALL construct when using UNDIGIT literals.

The declaration of USAGE IS INDEX data items subordinate to USAGE DISPLAY group items.

The ability to MOVE SPACES to DISPLAY and COMP-2 numeric data items.

The optional specification of an immediately-preceding redefining data item using the REDEFINES clause in the DATA DIVISION.

The recognition of a FILE CONTAINS clause in a File Description Entry.

The implicit qualification of a "DEPENDING ON" variable in an OCCURS DEPENDING clause by an 01-level record name.

The optional use of the reserved word THEN as a delimiter between a conditional expression and the first statement in an IF statement in the PROCEDURE DIVISION section.

The ability to use FILLER items as group items; explained in the DATA-NAME FILLER sub-section of the DATA DIVISION section.

The ZIP option of the CALL statement in the PROCEDURE DIVISION section.

D. COBOL SYNTAX SUMMARY

IDENTIFICATION DIVISION

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

[PROGRAM-ID.  comment-entry.]
[AUTHOR.  comment-entry.]
[DATE-COMPILED.  comment-entry.]
[DATE-WRITTEN.  comment-entry.]
[INSTALLATION.  comment-entry.]
[SECURITY.  comment-entry.]
```

ENVIRONMENT DIVISION

```
ENVIRONMENT DIVISION.
[CONFIGURATION SECTION.
 [SOURCE-COMPUTER.  comment entry.]
 [OBJECT-COMPUTER.  object-computer entry.]
 [SPECIAL-NAMES.  special-names entry.]]
[{I-O SECTION.            }
 {INPUT-OUTPUT SECTION.   }
FILE-CONTROL.  file-control entry.
[I-O-CONTROL.  input-output-control entry ].]
```

CONFIGURATION SECTION.

Option 1:

```
SOURCE-COMPUTER. COPY library-name
                 [FROM seq. no.][{THROUGH} seq.no.]
                                {THRU   }
                 [REPLACING word-1 BY text-1
                            [, word-2 BY text-2]...].
```

Option 2:

```
SOURCE-COMPUTER. comment-entry.
```

Option 1:

```
OBJECT-COMPUTER. COPY library-name
                 [FROM seq.no.] [{THROUGH} seq.no.]
                                {THRU   }
                 [REPLACING word-1 BY text-1
                            [, word-2 BY text-2]...].
```

ENVIRONMENT DIVISION (cont)

Option 2:

```
OBJECT-COMPUTER. { B-6700 }
                 { B-7700 }
                 { word-3 }

  [, MEMORY SIZE integer { CHARACTERS }]
                         { WORDS      }
                         { MODULES    }

  [, DISK SIZE integer { WORDS   }]
                       { MODULES }

  [, SEGMENT-LIMIT IS integer]

  [, [integer] hardware-name]... .
  [,  program collating SEQUENCE IS alphabet-name ] .
```

Option 1:

```
SPECIAL-NAMES. COPY library-name
                    [FROM seq. no.][{ THROUGH } seq. no.]
                                   { THRU    }
                    [REPLACING word-1 BY text-1
                                  [, word-2 BY text-2]...].
```

Option 2:

```
SPECIAL-NAMES.

  [CURRENCY SIGN IS literal-1]
  [, DECIMAL-POINT IS COMMA]
  [, hardware-name IS mnemonic-name]...
  [, CHANNEL integer IS mnemonic-name]...
  [, literal-2 IS mnemonic-name]... .
```

Option 1: ANSI 74 alphabet-name clause.

```
SPECIAL NAMES.
            [, alphabet-name IS { STANDARD-1 }]...
                                { NATIVE     }
                                { BCL        }
```

Option 2: ANSI 74 alphabet-name clause.

```
SPECIAL-NAMES.
    [                          literal-1 [{THROUGH}literal-2                          ]
    [,alphabet-name IS                   [{THRU   }                                   ]
    [                                    [ALSO literal-3 [,ALSO literal-4]...]       ]
    [                                                                                ]
    [                          [literal-5 [{THROUGH}literal-6                       ]]...
    [                                     [{THRU   }                                 ]
    [                                     [ALSO literal-7 [,ALSO literal-8]...]      ]
    [                                                                                ]
                                                                                        ...
I-O SECTION.
INPUT-OUTPUT SECTION.
```

Option 1:

```
FILE-CONTROL.  COPY library-name
                    [FROM seq. no.] [{THROUGH} seq. no.]
                                    {THRU   }
                    [REPLACING word-1 BY text-1
                               [, word-2 BY text-2]...].
```

Option 2:

```
FILE-CONTROL.

  SELECT [LOCAL | GLOBAL] [RECEIVED BY {REFERENCE | REF}] [OPTIONAL]  file-name

    ASSIGN TO [integer-1 [* integer-2]] [INTERCHANGE]

    {hardware-name | [DIRECT] hardware-name} [SINGLE] [BY {AREA | CYLINDER}] [SAVE]

    [; RESERVE {NO | integer-3 | data-name-1} [ALTERNATE [AREA | AREAS]]]

    [; {FILE-LIMIT IS | FILE-LIMITS ARE} {data-name-2 | literal-1} {THROUGH | THRU} {data-name-3 | literal-2 | END}]

    [; ACCESS MODE IS {SEQUENTIAL | RANDOM}]

    [; ACTUAL KEY IS data-name-4].
    [; ORGANIZATION IS SEQUENTIAL]
    [; FILE STATUS IS data-name-5].
```

Option 1:

```
I-O-CONTROL.  COPY library-name
                    [FROM seq. no.] [{THROUGH | THRU} seq. no.]
                    [REPLACING word-1 BY text-1
                              [, word-2 BY text-2] ... ].
```

Option 2:

```
I-O-CONTROL.

  [APPLY comment entry] ...

  [; MULTIPLE FILE TAPE CONTAINS [file-name-1 [POSITION integer-1]] ...]...

  [; SAME [RECORD | SORT] AREA FOR file-name-2 {, file-name-3} ...]...

  [; RERUN ON {DISK | DISKPACK} EVERY integer-2 RECORDS OF file-name-4] ... .
```

DATA DIVISION

```
DATA DIVISION.   [PREPARED FOR system-name.]
[ FILE SECTION.
    [ { file-description-entry          } [record-description-entry] ... ] ... ]
      { sort-merge-description-entry    }
[ DATA-BASE SECTION.
    [01 [internal-set-name] INVOKE set-name] ... ]
[ WORKING-STORAGE SECTION.
    [ 77-level-description-entry ] ... ]
      record-description-entry
[ CONSTANT SECTION.
    [ 77-level-description-entry ] ... ]
      record-description-entry
[ LINKAGE SECTION.
    [ 77-level-description-entry ] ... ]
      record-description-entry
[ LOCAL-STORAGE SECTION.

    [ LD local-storage-name.
        [ 77-level-description-entry ] ... ] ... ]
          record-description-entry
[REPORT SECTION.
    [report-description-entry {report-group-description-entry} ...] ...]
```

FILE SECTION.

```
Option 1:

{ FD } file-name COPY library-name
{ SD }

                    [ FROM seq. no.] [ { THROUGH } seq. no. ]
                                       { THRU    }
                    [ REPLACING word-1 BY text-1
                                [, word-2 BY text-2]...] .
```

Option 2:

```
FD file-name

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

[; BLOCK CONTAINS [integer-1 TO] integer-2 [ CHARACTERS ]]
                                           [ RECORDS    ]
                                           [ WORDS      ]

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

[; RECORD CONTAINS [integer-3 TO] integer-4 [ ASCII              ] [ CHARACTERS ]]
                                            [ COMP               ] [ WORDS      ]
                                            [ COMPUTATIONAL      ]
                                            [ COMP-2             ]
                                            [ COMPUTATIONAL-2    ]
                                            [ DISPLAY            ]
                                            [ DISPLAY-1          ]

[; LABEL { RECORD IS  } { STANDARD                          }]
         { RECORDS ARE} { OMITTED [WITH MULTIPLE AT END]    }
                        { data-name-5 [,data-name-6]...     }
[; DATA { RECORD IS  } data-name-7 [, data-name-8]...]
        { RECORDS ARE}
[; { REPORT IS  } report-name-1 [, report-name-2]...]
   { REPORTS ARE}
[; LINAGE IS { integer-5   } LINES]
             { data-name-9 }
[; SAVE-FACTOR IS { integer-6   }]
                  { data-name-10}

[; { VA     } OF { ID             } IS { data-name-11 }]
   { VALUE  }      { IDENTIFICATION }   { literal-1   }
   { VALUES }

[; CODE-SET IS alphabet-name].
```

Option 3:

```
SD file-name

[; RECORD CONTAINS integer-4 [ CHARACTERS ]]
                             [ WORDS      ]

[; DATA { RECORD IS  } data-name-7 [, data-name-8]...].
        { RECORDS ARE}
```

Option 1:

```
01  data-name-1; COPY  library-name
    [FROM seq. no.][{THROUGH | THRU} seq. no.]
        [REPLACING word-1 BY text-1
            [, word-2 BY text-2]...] .
```

Option 2:

```
level-number {data-name-1 | FILLER} [; REDEFINES data-name-2]
[; SEGMENT]
[;{SIZE | SZ} IS [integer-1 TO] integer-2 CHARACTERS [DEPENDING ON data-name-3]]
[;{PICTURE | PIC} IS character-string [DEPENDING ON data-name-4]]
[; GLOBAL]
[; LOCAL]
[; OWN]
[;[USAGE IS] {COMP                  {ASCII
              COMPUTATIONAL          DISPLAY
              COMP-1                 DISPLAY-1
              COMPUTATIONAL-1        INDEX
              COMP-2                 EVENT
              COMPUTATIONAL-2        LOCK
              COMP-4                 CONTROL-POINT
              COMPUTATIONAL-4        CP
              COMP-5
              COMPUTATIONAL-5
              INDEX FILE [CONTAINS file-name-1 [, file-name-2]...]}]
[; {OC | OCCURS} [integer-3 TO] integer-4 TIMES [DEPENDING ON data-name-4]
    [{ASCENDING | DESCENDING} KEY IS data-name-5 [, data-name-6]...] ...
    [INDEXED BY index-name-1 [, index-name-2]...]]
```

Option 2 (cont):

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

[; { JUSTIFIED } RIGHT]
   { JUST      }

[; RANGE IS literal-1 { THROUGH } literal-2]
                      { THRU    }

[; RECEIVED BY { REFERENCE }]
               { REF       }
               { CONTENT   }

[; BLANK WHEN ZERO]

[; { VA     }[IS ]
   { VALUE  }[ARE] literal-1]
   { VALUES }

[; RECORD AREA]

[; WITH { LOWER-BOUNDS }].
        { LOWER-BOUND  }
```

Option 3:

```
66 data-name-1 ; RENAMES data-name-2 [{ THROUGH } data-name-3 ].
                                      { THRU    }
```

Option 4:

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

[, literal-3 [{ THROUGH } literal-4 ]] ... .
              { THRU    }
```

REPORT SECTION

Option 1:

```
RD report-name; COPY library-name
                    [ FROM seq. no.][ { THROUGH } seq. no.]
                                     { THRU    }
                    [ REPLACING word-1 BY text-1
                                  [, word-2 BY text-2] ...] .
```

Option 2:

```
RD report-name
  [; CODE literal-1]
  [; { CONTROL IS  } { FINAL                                    } ]
     { CONTROLS ARE} { FINAL, data-name-1 [, data-name-2]...    }
                     { data-name-1 [, data-name-2]...           }
  [; PAGE [LIMIT IS ] integer-1 [LINE ]
          [LIMITS ARE]          [LINES]
      [, HEADING integer-2]
      [, FIRST DETAIL integer-3]
      [, LAST DETAIL integer-4]
      [, FOOTING integer-5] ] .
```

Option 1:

```
01 [data-name-1] ; COPY library-name
  [ FROM seq. no.][ { THROUGH } seq. no.]
                    { THRU    }
  [ REPLACING word-1 BY text-1
      [, word-2 BY text-2]...] .
```

```
Option 2:

01 [data-name-1]

[; LINE NUMBER IS { integer-1 ON NEXT PAGE }]
                  { PLUS integer-2         }

[; NEXT GROUP IS { integer-3       }]
                 { PLUS integer-4  }
                 { NEXT PAGE       }

                 { { RH              }                          }
                 { { REPORT HEADING  }                          }
                 {                                              }
                 { { PH              }                          }
                 { { PAGE HEADING    }                          }
                 {                                              }
                 { { CH              }  { data-name-2 }         }
                 { { CONTROL HEADING }  { FINAL       }         }
                 {                                              }
 ; TYPE IS       { { DE     }                                   }
                 { { DETAIL }                                   }
                 {                                              }
                 { { CF              }  { data-name-3 }         }
                 { { CONTROL FOOTING }  { FINAL       }         }
                 {                                              }
                 { { PF              }                          }
                 { { PAGE FOOTING    }                          }
                 {                                              }
                 { { RF              }                          }
                 { { REPORT FOOTING  }                          }

[; [USAGE IS] { DISPLAY   }] .
              { DISPLAY-1 }
```

```
Option 3:

level-number [data-name-1]

  [; LINE NUMBER IS { integer-1 [ON NEXT PAGE] }]
                    { PLUS integer-2           }

  [; [USAGE IS] { DISPLAY   }] .
                { DISPLAY-1 }
```

```
Option 4.
level-number [data-name-1]

  [; BLANK WHEN ZERO]
  [; COLUMN NUMBER IS integer-3]
  [; GROUP INDICATE]
```

Option 4 (cont):

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

[;  LINE NUMBER IS { integer-1 ON NEXT PAGE }]
                   { PLUS integer-2         }

  ; { PICTURE } IS character-string
    { PIC     }

{;  SOURCE IS { TODAYS-DATE }                                    }
{              { identifier-1 }                                  }
{                                                                }
{; {SUM identifier-2 [, identifier-3] ...                        }
{     [UPON data-name-2 [, data-name-3] ...] ... }...            }
{     [ RESET ON { FINAL       }]                                }
{                { data-name-4 }                                 }
{                                                                }
{; { VA    } is literal-1                                        }
{  { VALUE }                                                     }

[; USAGE IS] { DISPLAY   } .
             { DISPLAY-1 }
```

PROCEDURE DIVISION

```
PROCEDURE DIVISION [ USING { data-name         } ... ] .
                           { file-name         }
                           { control-point-name}
                           { event-name        }
                           { lock-name         }

[MONITOR statement.]

[DUMP statement.]...

[DECLARATIVES.

 section-name SECTION.  declarative-statement.

 paragraph-name.  [statement.]...

[paragraph-name.  [statement.]...] ...

[section-name SECTION.  declarative-statement.

 paragraph-name.  [statement.]...

[paragraph-name.  [statement.]...] ...] ...

END DECLARATIVES.]

[[section-name SECTION [priority-number].]

  paragraph-name.

[[paragraph-name.] ...  [statement.]...] ...] ...
```

Verb Formats

```
Option 1:
ACCEPT identifier [ FROM { hardware-name } ]
                         { mnemonic-name }
Option 2:
ACCEPT identifier  FROM { DATE }
                        { DAY  }
                        { TIME }
```

```
Option 1:

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

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

      [; ON SIZE ERROR statement [ELSE statement]]
```

Option 2:

```
ADD { identifier-1 } , { identifier-2 } [ , { identifier-3 } ]   GIVING
    { literal-1    }   { literal-2    }     { literal-3    }
      identifier-m [ROUNDED] [, identifier-n [ROUNDED]] ...
    [; ON SIZE ERROR statement [ELSE statement]]
```

Option 3:

```
ADD { CORR          } identifier-1 TO identifier-2 [ROUNDED]
    { CORRESPONDING }
    [; ON SIZE ERROR statement [ELSE statement]]
```

```
ALLOW { section-name-1 [, section-name-2]... }
      { INTERRUPT                            }
```

```
ALTER procedure-name-1 TO [PROCEED TO] procedure-name-2
    [, procedure-name-3 TO [PROCEED TO] procedure-name-4] ...
```

```
ATTACH section-name TO event-identifier
```

Option 1:

```
AWAIT { control-point-identifier ([subscript,] EXCEPTIONEVENT) }
      { (formula)                                              }
      { event-identifier                                       }
      { INTERRUPT                                              }
```

Option 2:

```
AWAIT area-identifier [ON EXCEPTION statement [ELSE statement]]
```

Option 1:

```
CALL control-point-identifier [WITH section-name]
     [USING actual-parameter-list]
```

Option 2:

```
CALL { section-name              } [USING actual-parameter-list]
     { installation-intrinsic    }
```

Option 3:

```
CALL PROGRAM DUMP
```

Option 4:

```
{ CALL SYSTEM WITH } data-name
{      ZIP         } file-name
```

PROCEDURE DIVISION (cont)

```
CAUSE [AND RESET] event-identifier-1 [ , event-identifier-2] ...
```

```
CHECKPOINT
      [TO   {DISKPACK}]
            {DISK    }
      [WITH {PURGE  }]
            {LOCK   }
      [; ON EXCEPTION {statement     } [ELSE {statement     }]]
                      {NEXT SENTENCE}       {NEXT SENTENCE}
```

Option 1:

```
CLOSE file-name-1 [WITH {NO REWIND}]
                       {LOCK     }
                       {RELEASE  }
                       {PURGE    }
                       {CRUNCH   }
  [, file-name-2 [WITH {NO REWIND}]]...
                       {LOCK     }
                       {RELEASE  }
                       {PURGE    }
                       {CRUNCH   }
```

Option 2:

```
CLOSE file-name-1 {REEL} [WITH NO REWIND]
                  {UNIT} [FOR REMOVAL   ]
     [,file-name-2 {REEL} [WITH NO REWIND]
                   {UNIT} [FOR REMOVAL   ]
```

Option 3:

```
CLOSE HERE file-name [WITH NO REWIND]
```

```
COMPUTE identifier-1 [ROUNDED] [, identifier-2 [ROUNDED]] ...

  {FROM  } {identifier-n}
  {=     } {literal-1   }
  {EQUALS} {formula     }

  [; ON SIZE ERROR statement [ELSE statement]]
```

```
CONTINUE control-point-identifier
```

```
COPY library-name [FROM seq. no.] [{THROUGH} seq. no.]
                                   {THRU   }
  [REPLACING word-1 BY text-1 [, word-2 BY text-2]...] .
```

PROCEDURE DIVISION (cont)

```
DEALLOCATE record-name

DETACH identifier-1 [, identifier-2]...

DISALLOW { section-name-1 [, section-name-2]... }
         { INTERRUPT                            }

DISPLAY { literal-1    } [, { literal-2    }] ... [UPON { mnemonic-name }]
        { identifier-1 }    { identifier-2 }           { hardware-name }

Option 1:

   DIVIDE [MOD] { literal-1    } INTO
                { identifier-1 }
          identifier-2 [ROUNDED] ,[identifier-3 [ROUNDED]] ...
          [; ON SIZE ERROR statement [ELSE statement]]

Option 2:

   DIVIDE [MOD] { literal-1    } INTO { literal-2    }
                { identifier-1 }      { identifier-2 }
          GIVING identifier-3 [ROUNDED][, identifier-4 [ROUNDED]] ...
          [; ON SIZE ERROR statement [ELSE statement]]

Option 3:

   DIVIDE [MOD] { literal-1    } BY { literal-2    }
                { identifier-1 }    { identifier-2 }
          GIVING identifier-3 [ROUNDED] [, identifier-4 [ROUNDED]] ...
          [; ON SIZE ERROR statement [ELSE statement]]

Option 4

   DIVIDE { literal    } INTO { literal-2    }
          { identifier-1 }    { identifier-2 }
          GIVING identifier-3 [ROUNDED] REMAINDER
          identifier-4 [ROUNDED] [;ON SIZE ERROR statement [ELSE
          statement]]

Option 5:

   DIVIDE { literal-1    } BY { literal-2    }
          { identifier-1 }    { identifier-2 }
          GIVING identifier-3 [ROUNDED] REMAINDER
          identifier-4 [ROUNDED] [;ON SIZE ERROR statement [ELSE
          statement]]

DUMP { file-name } _( { data-name-1    } [, { data-name-2    }] ... )_
     { PRINTER  }     { procedure-name }     { procedure-name }
                      { ALL            }     { ALL            }

   paragraph-name-n . { literal    } .
                      { data-name-n }
```

PROCEDURE DIVISION (cont)

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

Option 1:

```
EXAMINE identifier-1 TALLYING { ALL       } { literal-1    }
                              { LEADING   } { identifier-2 }
                              { UNTIL FIRST }

   [ REPLACING BY { literal-2    } ]
                  { identifier-3 }
```

Option 2:

```
EXAMINE identifier-1 REPLACING { ALL       }
                               { LEADING   }
                               { [UNTIL] FIRST }

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

```
EXECUTE control-point-identifier WITH section-name
   [ USING arithmetic-expression-1 [ , arithmetic-expression-2 ] ... ]
```

```
EXIT [ PROGRAM [RETURN HERE] ]
     [ PROCEDURE             ]
     [ HERE                  ] .
```

```
GENERATE identifier
```

Option 1:

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

Option 2:

```
GO TO procedure-name-1 [, procedure-name-2]...

  , procedure-name-n DEPENDING ON { formula    }
                                  { identifier }
```

```
IF condition-1; [THEN] { statement-1     } [ ; ELSE { statement-2     } ]
                       { NEXT SENTENCE }           { NEXT SENTENCE }
```

```
INITIATE report-name-1 [, report-name-2]...
```

```
LOCK { data-name       } [; AT LOCKED statement [ELSE statement]]
     { lock-identifier }
     { event-identifier }
```

```
MERGE file-name-1 ON { ASCENDING  } KEY data-name-1 [, data-name-2]...
                     { DESCENDING }

    [, ON { ASCENDING  } KEY data-name-3 [, data-name-4]...] ...
          { DESCENDING }

       USING file-name-2 [, file-name-3]... , file-name-9

    { GIVING file-name-9                                                    }
    { OUTPUT PROCEDURE IS section-name-1 [ { THROUGH } section-name-2 ]     }
                                           { THRU    }
```

```
MONITOR { file-name } ( { data-name-1      } [ , { data-name-2      } ]... )
        { PRINTER   }  { procedure-name-1 }      { procedure-name-2 }
                       { ALL              }      { ALL              }
```

```
Option 1:
MOVE { speical-register      } TO identifier-2 [, identifier-3]...
     { attribute-identifier  }
     { identifier-1          }
     { literal-1             }
```

```
Option 2:

MOVE { CORR          } identifier-1 TO identifier-2 [, identifier-3]...
     { CORRESPONDING }
```

```
Option 3:

MOVE [identifier-1 TO identifier-2]
```

```
Option 4:

MOVE identifier-1 TO identifier-2

    [ { literal-2 } + { literal-3 } + { literal-4 } ]
      { formula-1 }   { formula-2 }   { formula-3 }
```

Option 1:

```
MULTIPLY { literal-1    } BY
         { identifier-1 }

    identifier-2 [ROUNDED] [, identifier-3 [ROUNDED]] ...

    [; ON SIZE ERROR statement [ELSE statement]]
```

Option 2:

```
MULTIPLY { literal-1    } BY { literal-2    }
         { identifier-1 }    { identifier-2 }

    GIVING identifier-3 [ROUNDED] [, identifier-4 [ROUNDED]] ...

    [; ON SIZE ERROR statement [ELSE statement]]
```

Option 1:

```
                 { INPUT  file-name-1 [ {WITH LOCK [ACCESS]} ]                          }
                 {                    [ {REVERSED         } ]                          }
                 {                    [ {WITH NO REWIND   } ]                          }
                 {      [file-name-2  [ {WITH LOCK [ACCESS]} ]]                       }
                 {                    [ {REVERSED         } ]]...                     }
                 {                    [ {WITH NO REWIND   } ]]                        }
OPEN             { OUTPUT file-name-3   [WITH NO REWIND]                               }
                 {        [, file-name-4 [WITH NO REWIND]] ...                         }
                 { {I-O          }                                                     }
                 { {INPUT-OUTPUT } file-name-5 [, file-name-6] ...                     }
                 { {O-I          }                                                     }
                 { EXTEND         file-name-7 [, file-name-8] ...                      }
```

Option 2:

```
OPEN { INPUT  } file-name-7 REEL-NUMBER { literal    }
     { OUTPUT }                         { data-name }
```

Option 1:

```
PERFORM procedure-name-1 [ { THRU    } procedure-name-2 ]
                         [ { THROUGH } ]
```

PROCEDURE DIVISION (cont)

Option 2:

```
PERFORM procedure-name-1 [{THRU    } procedure-name-2]
                         {THROUGH  }

        {identifier-7}
        {integer-1   } TIMES
        {formula-1   }
```

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  }

        VARYING {identifier-1} FROM {identifier-2}
                {index-name-1}      {literal-2   }
                                    {formula-2   }
                                    {index-name-2}

        BY {identifier-3} UNTIL condition-1
           {literal-3   }
           {formula-3   }

      [ AFTER {identifier-4} FROM {identifier-5}
              {index-name-4}      {literal-5   }
                                  {formula-5   }
                                  {index-name-5}

        BY {identifier-6} UNTIL condition-2 ] ...
           {literal-6   }
           {formula-6   }
```

```
PROCESS control-point-identifier WITH section-name

      [ USING actual-parameter-list ]
```

Option 1:

```
READ file-name RECORD [INTO identifier]

  ; AT END statement [ELSE statement]
```

PROCEDURE DIVISION (cont)

```
Option 2:

READ file-name RECORD [INTO identifier]

    ; INVALID KEY statement [ELSE statement]
```

```
Option 3:

READ file-name [KEY IS formula]  INTO identifier

    [ USING event-identifier]

    [ON EXCEPTION statement [ELSE statement]]
```

```
RELEASE record-name [FROM identifier]
```

```
RESET event-identifier-1 [, event-identifier-2]...
```

```
RETURN file-name RECORD [INTO identifier]

    ; AT END statement [ELSE statement]
```

```
RUN control-point-identifier WITH section-name

    [ USING   arithmetic-expression-1  [, arithmetic-expression-2 ] ...]
```

```
SEARCH identifier-1 [ALL][VARYING  {identifier-2 }]
                           {index-name-1}

   [; AT END imperative-statement-1]

    ; WHEN condition-1  {imperative-statement-2}
                        {NEXT SENTENCE        }

   [; WHEN condition-2  {imperative-statement-3}]  ...
                        {NEXT SENTENCE        }
```

```
SEEK file-name RECORD [WITH KEY CONVERSION] ...
```

```
Option 1:

SET  {identifier-1}  [,  {identifier-2}]  ...  TO  {identifier-3}
     {index-name-1}      {index-name-2}            {index-name-3}
                                                   {integer-1    }
```

Option 2:

```
SET index-name-4 [, index-name-5]... { UP BY   }  {identifier-4}
                                     { DOWN BY }  {integer-2   }
```

Option 3 (FILE and BUFFER ATTRIBUTES):

```
SET { buffer-name }
    { file-name-1 }  ( attribute-expression )
    { FILE        }

    { TO      }  {identifier-4          }
    { UP BY   }  {literal-2             }
    { DOWN BY }  {formula-2             }
                 {VALUE attribute-mnemonic}
```

Option 4 (TASKING):

```
SET { MYJOB                  }  ( attribute-expression )
    { MYSELF                 }
    { control-point-identifier }

    TO {identifier-4          }
       {literal-2             }
       {formula-2             }
       {VALUE attribute-mnemonic}
```

```
SORT file-name-1 [ {PURGE} ON ERROR ] ON {ASCENDING } KEY data-name-1 [,data-name-2].
                 {RUN   }               {DESCENDING}
                 {END   }
   [,ON {ASCENDING } KEY data-name-3 [, data-name-4] ...] ...
        {DESCENDING}
   { USING  file-name-2 [ {LOCK   } ] }
   {                    {PURGE  } }
   {                    {RELEASE} }
   { INPUT PROCEDURE IS section-name-1 [ {THROUGH} section-name-2 ] }
   {                                    {THRU   }                  }
   { GIVING  file-name-3 [ {LOCK   } ] }
   {                     {RELEASE} }
   { OUTPUT PROCEDURE IS  section-name-3 [ {THROUGH} section-name-4 ] }
   {                                      {THRU   }                  }
   [ MEMORY SIZE  formula-1 {CHARACTERS} ]
   [                        {WORDS     } ]
   [                        {MODULES   } ]
   [ DISK SIZE  formula-2 {WORDS  } ]
   [                      {MODULES} ]
   [ RESTART  IS {formula-3 } ]
   [             {data-name-5}
   [             {literal-1 } ]
```

Option 1:

```
STOP RUN
```

Option 2:

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

Option 1:

```
SUBTRACT {literal-1   } [, {literal-2   }] ... FROM
         {identifier-1}    {identifier-2}
    identifier-m [ROUNDED] [, identifier-n [ROUNDED]] ...
    [; ON SIZE ERROR statement [ELSE statement]]
```

Option 2:

```
SUBTRACT {literal-1   } [, {literal-2   }] ...
         {identifier-1}    {identifier-2}
    FROM {literal-m   } GIVING
         {identifier-m}
       identifier-n [ROUNDED] [, identifier-o [ROUNDED]] ...
    [; ON SIZE ERROR statement [ELSE statement]]
```

Option 3:

```
SUBTRACT {CORR         } identifier-1
         {CORRESPONDING}
      FROM identifier-2 [ROUNDED]
    [; ON SIZE ERROR statement [ELSE statement]]
```

```
TERMINATE report-name-1 [, report-name-2]...
```

```
UNLOCK {identifier      }...
       {lock-identifier }
       {event-identifier}
```

Option 1:

```
USE AFTER {RECORD SIZE ERROR          }
          {STANDARD ERROR PROCEDURE   }
    ON {INPUT-OUTPUT                          }.
       {I-O                                   }
       {INPUT                                 }
       {file-name-1 [, file-name-2]...        }
```

PROCEDURE DIVISION (cont)

Option 2:

```
USE { BEFORE } STANDARD { BEGINNING } { REEL }
    { AFTER  }          { ENDING    } { FILE }

    LABEL PROCEDURE ON { INPUT-OUTPUT                    }.
                       { I-O                             }
                       { INPUT                           }
                       { OUTPUT                          }
                       { file-name-1 [, file-name-2]...  }
```

Option 3:

```
USE { EXTERNAL { identifier-1   } AS PROCEDURE }
    { AS GLOBAL { mnemonic-name }              }

  [ ; WITH { local-file-name    } [ , { local-file-name    } ] ... ]
           { local-storage-name }     { local-storage-name }
    ; USING identifier-2 [, identifier-3]... ] .
```

Option 4:

```
USE AS INTERRUPT PROCEDURE.
```

Option 5:

```
USE AFTER STANDARD { EXCEPTION } PROCEDURE ON { file-name-1 [,file-name-2]... }
                   { ERROR     }              { INPUT                          }
                                              { OUTPUT                         }
                                              { I-O                            }
                                              { EXTEND                         }
```

Option 1:

```
WAIT { formula          }
     { event-identifier }
     { INTERRUPT        }
```

Option 2:

```
WAIT control-point-identifier ([subscript,] EXCEPTIONEVENT)
```

Option 3:

```
WAIT [AND RESET] [formula]
   event-identifier-1 [, event-identifier-2]...
   [GIVING data-name-1]
```

Option 4:

```
WAIT area-identifier [ON EXCEPTION statement [ELSE statement]]
```

PROCEDURE DIVISION (cont)

Option 1:

```
WRITE record-name [FROM identifier-1]

    [ { BEFORE } ADVANCING TO { CHANNEL integer-1      } ]
    [ { AFTER  }              { integer-2 LINES        } ]
                              { identifier-2 LINES     }
                              { mnemonic-name          }
                              { PAGE                   }

    [ ; AT { END-OF-PAGE } statement [ELSE statement] ]
           { EOP         }
```

Option 2:

```
WRITE record-name [FROM identifier-1]
    ; INVALID KEY statement [ELSE statement]
```

Option 3:

```
WRITE record-name [FROM identifier-1] TO { AUXILIARY }
                                         { ALTERNATE }
                                         { ERROR     }
```

Option 4:

```
WRITE file-name [KEY IS formula]

    FROM identifier [USING event-identifier]

    [; ON EXCEPTION statement [ELSE statement]]
```

E. ANSI 74 COBOL SYNTAX SUMMARY

INDEXED I-O

```
INPUT-OUTPUT SECTION.
FILE-CONTROL.
SELECT file-name
    ASSIGN to [ integer-1 [* integer-2] ] { DISK     }
                                          { DISKPACK }
    [; RESERVE integer-3   [AREA ]]
                           [AREAS]
    ; ORGANIZATION IS INDEXED
    [; ACCESS MODE IS    {SEQUENTIAL}]
                         {RANDOM    }
                         {DYNAMIC   }
    ; RECORD KEY IS data-name-1
   [; FILE STATUS IS data-name-2]
```

```
  {VA    }
  {VALUE  OF {ID            } IS {data-name-1}
  {VALUES    {IDENTIFICATION}   {literal-1  }
          [, KEYSPERENTRY IS integer-1]
          [, AREAOVERFLOW IS integer-2]
          [, FILEOVERFLOW IS integer-3]
```

```
  CLOSE file-name-1 [WITH {LOCK }]
                         {PURGE}
      [, file-name-2 [WITH {LOCK }]]   ...
                           {PURGE}
```

INDEXED I-O (cont)

```
DELETE file-name RECORD [; INVALID KEY imperative-statement]

        OPEN { INPUT  file-name-1 [, file-name-2] ... }
             { OUTPUT file-name-3 [, file-name-4] ... } ...
             { I-O    file-name-5 [, file-name-6] ... }

Option 1:

        READ file-name [NEXT] RECORD [INTO identifier]
                [; AT END imperative-statement]

Option 2:

        READ file-name RECORD [INTO identifier]
             [; KEY IS data-name]
             [; INVALID KEY imperative-statement]

        REWRITE record-name [FROM identifier]
              [; INVALID KEY imperative-statement]

        START file-name [ KEY { IS EQUAL TO                } data-name ]
                              { IS =                       }
                              { IS GREATER THAN            }
                              { IS >                       }
                              { IS NOT LESS THAN           }
                              { IS NOT <                   }
            [; INVALID KEY imperative-statement]

        USE AFTER STANDARD { EXCEPTION } PROCEDURE ON { file-name-1[,file-name-2]...
                           { ERROR     }              { INPUT
                                                      { OUTPUT
                                                      { I-O

        WRITE record-name [FROM identifier][;INVALID KEY imperative-statement]
```

SEQUENTIAL I-O

```
LINAGE IS {data-name-1 | integer-1} LINES [,WITH FOOTING AT {data-name-2 | integer-2}]
  [,LINES AT TOP {data-name-3 | integer-3}][,LINES AT BOTTOM {data-name-4 | integer-4}]
```

```
WRITE record-name [FROM  identifier-1]
  [ {BEFORE | AFTER} ADVANCING { {identifier-2 | integer} [LINE | LINES] | {mnemonic-name | PAGE} } ]
  [; AT {END-OF-PAGE | EOP} imperative-statement ]
```

NUCLEUS

Option 1:

```
INSPECT identifier-1 TALLYING
{ , identifier-2 FOR { , { ALL | LEADING | CHARACTERS } { identifier-3 | literal-1 } } [ { BEFORE | AFTER } INITIAL { identifier-4 | literal-2 } ] } ... } ...
```

Option 2:

```
INSPECT identifier-1 REPLACING
{ CHARACTERS BY { identifier-6 | literal-4 } [ { BEFORE | AFTER } INITIAL { identifier-7 | literal-5 } ]
{ { , { ALL | LEADING | FIRST } } { , { identifier-5 | literal-3 } BY { identifier-6 | literal-4 } [ { BEFORE | AFTER } INITIAL { identifier-7 | literal-5 } ] } ... } ... }
```

Option 3:

```
INSPECT identifier-1 TALLYING
{ , identifier-2 FOR { , { ALL | LEADING | CHARACTERS } { identifier-3 | literal-1 } } [ { BEFORE | AFTER } INITIAL { identifier-4 | literal-2 } ] } ... } ...
REPLACING
{ CHARACTERS BY { identifier-6 | literal-4 } [ { BEFORE | AFTER } INITIAL { identifier-7 | literal-5 } ]
{ { { ALL | LEADING | FIRST } } { , { identifier-5 | literal-3 } BY { identifier-6 | literal-4 } [ { BEFORE | AFTER } INITIAL { identifier-7 | literal-5 } ] } ... } ... }
```

NUCLEUS (cont)

```
[SIGN IS] { LEADING  } [SEPARATE CHARACTER]
          { TRAILING }
```

```
STRING { identifier-1 } [ ,identifier-2 ] ... DELIMITED BY { identifier-3 }
       { literal-1    }   [ ,literal-2   ]                  { literal-3    }
                                                           { SIZE         }
    [ , { identifier-4 } [ ,identifier-5 ] ... DELIMITED BY { identifier-6 } ] ...
        { literal-4    }   [ ,literal-5   ]                  { literal-6    }
                                                            { SIZE         }
      INTO identifier-7  [WITH-POINTER identifier-8]
    [;  ON OVERFLOW imperative-statement]
```

```
UNSTRING identifier-1
    [ DELIMITED BY [ALL] { identifier-2 } , [ OR [ALL] { identifier-3 } ] ... ]
                         { literal-1    }               { literal-2    }
      INTO identifier-4 [, DELIMITER IN identifier-5][, COUNT IN identifier-6]

    [,  identifier-7 [,  DELIMITER IN identifier-8][, COUNT IN identifier-9] ...
    [WITH POINTER identifier-10][TALLYING IN identifier-11]
    [;  ON OVERFLOW  imperative-statement]
```

F. COMPILER PRODUCED MESSAGES

ERROR & WARNING MESSAGES

ERROR	MESSAGE
000	UNRECOGNIZED CONSTRUCT
001	IDENTIFIER EXCEEDS 30 CHARACTERS
002	STRING GREATER THAN 256 CHARACTERS
003	NUMBER > 23 DIGITS
004	INTEGER > 23 DIGITS
005	CANNOT START IN MARGIN A
006	STRING CONTINUATION INCORRECT
007	IMPROPER LIBRARY NAME
008	SEGMENT SIZE > 4095 WORDS
009	MEMORY SIZE MUST BE INTEGER
010	DISK SIZE MUST BE INTEGER OR NUMBER
011	MUST START IN MARGIN A
012	MERGE REQUIRES 1 TO 8 FILES
013	MUST BE INTEGER
014	AT LEAST 3 AND AT MOST 8 SORT-TAPES ARE ALLOWED
015	ILLEGAL HARDWARE TYPE
016	MUST BE SINGLE ALPHA CHARACTER
017	ILLEGAL FILE NAME
018	MUST BE DATA NAME
019	COMPILER ERROR
020	BLOCK SIZE MUST BE MULTIPLE OF RECORD SIZE
021	DIRECT INDEX FILES MUST BE LEVEL 77 ITEMS
022	ALL LABEL RECORDS MUST HAVE SAME USAGE
023	RERUN NOT ALLOWED ON SORT FILE
024	RERUN NOT ALLOWED ON DISK
025	FILE NOT SELECTED
026	DUPLICATE FD FOR THIS FILE
027	DUPLICATE OR INCOMPATIBLE CLAUSE
028	CLAUSE NOT LEGAL IN SD
029	INCORRECT CARD CONTINUATION
030	ILLEGAL USE OF RECORD AREA CLAUSE
031	FILE ALREADY ASSIGNED TO MULTI-FILE
032	ILLEGAL CLAUSE USED WITH SAVE ARRAY
033	MIN RECORD SIZE MUST BE LESS THAN MAX SIZE
034	MIN BLOCK SIZE MUST BE LESS THAN MAX SIZE
035	BLOCKED VARIABLE LENGTH RECORDS CANNOT BE RANDOM
036	NO GROUP ITEM MAY BE MOVED TO AN ELEMENTARY COMP
037	ALPHABETIC RECEIVING FIELD ILLEGAL
038	DISK AND REMOTE CANNOT BE UNLABELED
039	NON-NUMERIC LITERAL EXPECTED
040	INCORRECT FILE OR PROGRAM ID
041	SAVE FACTOR > 999

042 MUST BE LEVEL 1
043 FILLER MUST BE ELEMENTARY
044 SENDING ITEM IS NOT AN INTEGER
045 GROUP COMP RECEIVING ITEM ILLEGAL
046 QUALIFIER OR NAME HAS NOT APPEARED BEFORE
047 MISSING QUALIFICATION
048 QUALIFICATION FOUND NOTHING UNIQUE
049 QUALIFICATION IS NOT UNIQUE
050 NAME HAS NOT APPEARED BEFORE
051 RENAMED LEVEL NUMBER IS ILLEGAL
052 RENAMED ITEM CANNOT HAVE OCCURS
053 RENAMED CANNOT BE VARIABLE SIZE
054 RENAMES DN3 PRIOR TO DN2
055 RENAMES NOT NEXT TO RENAMED AREA
056 ILLEGAL DUPLICATE NAME
057 NESTED REDEFINES NOT ALLOWED
058 REDEFINES-CANNOT HAVE VALUE
059 REDEFINES-CANNOT BE VARIABLE SIZE
060 REDEFINES-CANNOT HAVE OCCURS
061 MUST BE ELEMENTARY ITEM
062 INVALID LEVEL NUMBER
063 SIZE ERROR-GROUP VS ELEMENTARY
064 REDEFINED AREA NOT SAME SIZE
065 REDEFINES NOT ADJACENT TO AREA
066 SECTION REQUIRED AFTER DECLARATIVES
067 PICTURE STRING ERROR
068 RIGHT PARENTHESIS EXPECTED
069 OPERAND EXPECTED
070 SIGN CLAUSE MUST BE USED WITH SIGNED NUMERIC ITEM
071 PICTURE STRING EXCEEDS 30 CHARACTERS
072 ALPHABET-NAME REQUIRED
073 ILLEGAL EXPRESSION STRUCTURE
074 FROM, = , OR EQUALS EXPECTED
075 DATA-NAME EXPECTED
076 STATEMENT CANNOT BEGIN WITH THIS ITEM
077 INFLEXIBLE COMPILER LIMIT EXCEEDED
078 TO EXPECTED
079 PERIOD REQUIRED AFTER PARAGRAPH NAME
080 PERIOD REQUIRED AFTER SECTION HEADING
081 MISSING PERIOD
082 USE STATEMENT EXPECTED
083 PARAGRAPH NAME REQUIRED
084 SECTION NAME REQUIRED
085 PERIOD REQUIRED AFTER DECLARATIVES HEADING
086 SOURCE INPUT SEQUENCE ERROR
087 CODE-SET REQUIRES A SIGN IS SEPARATE CLAUSE
088 ELEMENTARY ITEM MUST HAVE SIZE
089 SCALE EXCEEDS 23 DIGITS
090 NUMERIC ITEM EXCEEDS 23 DIGITS
091 CLASS CONFLICT
092 USAGE CONFLICT
093 J OR S NOT ALLOWED IN EDITED NUMERIC
094 CANNOT BE VARIABLE SIZE
095 COMPUTATIONAL REQUIRES NUMERIC CLASS
096 FILLER ADDED
097 ONLY ONE TYPE OF FLOAT/INSERT ALLOWED
098 MULTIPLE SIGNS SPECIFIED IN PICTURE
099 ALPHANUMERIC SENDING FIELD REQUIRED
100 INVALID OPTION ELEMENT

101 SUBSCRIPTS REQUIRED
102 ILLEGAL SUBSCRIPT
103 NOT ENOUGH SUBSCRIPTS
104 INDEX-NAMES AND DATA-NAMES MAY NOT BE MIXED
105 GIVING CLAUSE ILLEGAL IN THIS CONTEXT
106 FROM EXPECTED
107 GIVING EXPECTED
108 INTO OR BY EXPECTED
109 ELSE EXPECTED
110 LABEL REQUIRED
111 DEPENDING EXPECTED
112 LOCK MAY ONLY BE USED WITH OPEN INPUT OR I-O
113 CONSTRUCT NOT IMPLEMENTED
114 LITERAL VALUE EXPECTED
115 CONFLICTING CLASS
116 ILLEGAL FOR FILLER ITEM
117 LITERAL SIZE EXCEEDS DECLARED SIZE
118 COPY MUST BE ON 01 LEVEL
119 TIMES EXPECTED
120 SOURCE MUST BE SAME LENGTH AS TARGET
121 END OF STATEMENT EXPECTED
122 NON-ALPHABETIC CHARACTERS NOT ALLOWED IN STRING
123 VALUE CLAUSE EXPECTED
124 REDEFINES NOT ALLOWED FOR FILE 01 LEVELS
125 REDEFINED AREA TOO LARGE
126 CANNOT CHAIN REDEFINES
127 VALUE NOT ALLOWED IN FILE SECTION
128 VALUE SPECIFIED IN PRIOR LEVEL
129 TOO MANY LEFT PARENTHESES
130 CONSTANT SECTION REQUIRES VALUE
131 VALUE NOT ALLOWED
132 SERIALNO ONLY ALLOWED IN MOVE STATEMENTS
133 VALUE NOT ALLOWED FOR ITEM WITHIN OCCURS
134 USAGE CONFLICT BETWEEN LEVELS
135 ILLEGAL CLAUSE USED WITH INDEX USAGE
136 NON-DATA ITEMS CANNOT HAVE CONDITION-NAMES
137 ILLEGAL COMPARISON OF TWO OPERANDS
138 ILLEGAL LOGICAL OPERATION
139 MISSING SUBJECT OF RELATION
140 LEFT PARENTHESIS EXPECTED
141 ILLEGAL USE OF LOGICAL CONNECTIVE
142 RELATIVE SINGLE SPACING ASSUMED
143 CONDITION DID NOT RESULT IN TRUTH VALUE
144 BOOLEAN FUNCTION ASSUMED TO BE INTEGER OPERAND
145 AT END CLAUSE EXPECTED
146 INVALID KEY CLAUSE EXPECTED
147 STATISTICS NOT IMPLEMENTED ABOVE LEVEL 2
148 MORE THAN 3 DIMENSIONS
149 INDEX NAME MUST BE UNIQUE
150 JUSTIFIED NOT ALLOWED
151 RECEIVING FIELD MAY NOT BE SCALED OR EDITED
152 ALPHANUMERIC OPERAND EXPECTED
153 UNDIGIT LITERALS ALLOWED ONLY WITH 4-BIT ITEMS
154 NO DATA RECORD DESCRIPTION
155 TRUNCATION OF NON-ZERO DIGITS
156 PAGE LIMIT CLAUSE REQUIRED IN RD ENTRY
157 NO FILE HAS LINAGE CLAUSE
158 ILLEGAL LEVEL NUMBER WITH OCCURS CLAUSE
159 MISSING LABEL

160 LITERAL LENGTH SHOULD BE ONE, 1ST CHARACTER USED
161 FIRST EXPECTED
162 DUPLICATE LABEL
163 ALTER VERB - INCORRECT USAGE
164 MUST BE NUMERIC VALUE
165 ILLEGAL ARITHMETIC STATEMENT STRUCTURE
166 NO CORRESPONDING ITEMS
167 TO OR UP BY OR DOWN BY EXPECTED
168 ITEM CANNOT BE ZERO SIZE
169 J NOT ALLOWED FOR COMP
170 ILLEGAL KEY ITEM
171 IDENTIFIER MUST HAVE OCCURS CLAUSE
172 IDENTIFIER MUST HAVE INDEXED BY CLAUSE
173 WHEN PHRASE REQUIRED
174 IDENTIFIER SHOULD HAVE KEY CLAUSE
175 KEY MUST BE USED IN CONDITION
176 DIAGNOSTIC MUST BE A UNIQUE NAME
177 MUST BE ASSIGNED TO MERGE
178 IDENTIFIER MAY NOT BE MONITORED
179 TASK OPERAND EXPECTED
180 EXIT MUST BE ONLY STATEMENT IN PARAGRAPH
181 FLOATING POINT QUOTIENT NOT ALLOWED
182 IMPROPER WORD FORMAT
183 ILLEGAL CODE LITERAL OR COLUMN NUMBER
184 NAME MUST BE UNIQUE
185 MUST BE REPORT NAME
186 ILLEGAL LINE VALUE OR LINE CLAUSE OMITTED
187 MUST BE DETAIL
188 ILLEGAL CHANNEL VALUE
189 REPORT ENTRY MUST CONTAIN REPORT GROUP
190 THIS REPORT FAILS TO SPECIFY A CODE LITERAL
191 NEXT PAGE EXPECTED
192 MAY ONLY APPEAR AT LEVEL 1
193 GROUP EXPECTED
194 OF EXPECTED
195 CONTROL LEVEL EXPECTED
196 ILLEGAL CONTROL LEVEL
197 FOOTING OR HEADING
198 ILLEGAL DUPLICATE TYPE GROUP
199 ILLEGAL GROUP TYPE
200 BOOLEAN OPERAND EXPECTED
201 FILE NOT ASSIGNED TO TAPE OR DISK
202 ROUTINE NOT APPLICABLE TO ANY FILE
203 FILE NOT ASSIGNED TO DISK
204 NOT A LABELED FILE
205 FILE NOT ASSIGNED TO TAPE
206 STRINGS MAY NOT BE MOVED TO COMP OR COMP-1 ITEMS
207 ILLEGAL RECORD NAME
208 CHANNEL OR LINE NUMBER EXPECTED
209 MUST BE ASSIGNED TO SORT
210 FLOATING POINT RECEIVING FIELD OPERAND EXPECTED
211 ILLEGAL CLOSE ACTION
212 ILLEGAL OPEN ACTION
213 ONLY FIRST 256 CHARACTERS WILL BE DISPLAYED
214 REPORT-NAME OR DETAIL-GROUP-NAME EXPECTED
215 REPORT-NAME OR ALL EXPECTED
216 MUST BE RECORD NAME OF SD
217 MUST BE SORT FILE NAME
218 KEY ILLEGAL OR IMPROPER QUALIFICATION

```
219       ILLEGAL SORT KEY
220       NO SORT KEYS SPECIFIED
221       INPUT PROCEDURE NOT ALLOWED IN MERGE
222       ONLY ONE FILE ALLOWED
223       MUST BE A TAPE FILE
224       ARE YOU SURE YOU WANT A TWO-DIMENSIONAL ARRAY?
225       MUST BE DATA BASE OR PARTITION NAME
226       ILLEGAL SEEK ACTION
227       EVERY TABLE DIMENSION MUST HAVE INDEXED CLAUSE
228       ILLEGAL REFERENCE TO CONSTANT
229       POSSIBLE INTEGER OVERFLOW
230       SORT PROCEDURE MUST BE CONTIGUOUS
231       SORT PROCEDURE MUST NOT CONTAIN SORT VERB
232       RELEASE MUST APPEAR IN INPUT PROCEDURE
233       RETURN MUST APPEAR IN OUTPUT PROCEDURE
234       INVALID TRANSFER OF CONTROL
235       INVALID OVERLAPPING OR NONCONTIGUOUS PROCEDURE
236       ILLEGAL DATA STRUCTURE FOR CORRESPONDING
237       # OF DECIMAL PLACES MOVED IS FUNCTION DEPENDENT
238       LITERAL MUST BE GREATER THAN PREDECESSOR
239       COMPILER MUST HAVE EXCLUSIVE USE OF CODE FILE
240       OPERAND COMPOSITE SIZE ERROR
241       RECORD KEY MUST RESIDE WITHIN RECORD AREA
242       NUMERIC LITERAL EXCEEDS SPECIFIED LIMITS
243       ILLEGAL USE OF FIGURATIVE
244       INCOMPLETE SORT PROCEDURE
245       DATA NAME OPERAND EXPECTED
246       LITERAL OPERAND EXPECTED
247       ARITHMETIC OPERAND EXPECTED
248       MOVE SENDING FIELD OPERAND EXPECTED
249       SET RECEIVING FIELD OPERAND EXPECTED
250       SET SENDING FIELD OPERAND EXPECTED
251       NUMERIC DATA NAME OPERAND EXPECTED
252       NUMERIC LITERAL OPERAND EXPECTED
253       NUMERIC SENDING FIELD OPERAND EXPECTED
254       NUMERIC RECEIVING FIELD OPERAND EXPECTED
255       FIGURATIVE OR INTRINSIC OPERAND EXPECTED
256       DUPLICATE LABEL RECORD DESCRIPTION
257       MISSING LABEL RECORD DECLARATION
258       LABEL RECORDS MUST BE 80 CHARACTERS
259       MORE THAN 9 LABEL RECORDS SPECIFIED
260       RANDOM DISK FILE MUST HAVE ACTUAL KEY
261       MUST BE ELEMENTARY CONTROL-POINT IDENTIFIER
262       RECORD DESCRIPTION NOT NECESSARY FOR DIRECT FILE
263       SIZE OF ALL RECORDS MUST DEPEND ON SAME ITEM
264       REPLACING TABLE LIMIT EXCEEDED
265       MISSING BY IN REPLACING STATEMENT
266       INDEXED FILE MUST HAVE RECORD KEY CLAUSE
267       NO LIBRARY SEQ NUM = OR > THAN STARTING SEQUENCE
268       INVALID COPY TERMINUS
269       MNEMONIC VALUE EXPECTED
270       FILE ATTRIBUTE EXPECTED
271       INVALID SET STRING
272       PERFORM RANGE CONFLICT
273       INVALID USE OF GENERALIZED FILE SPECIFIER
274       INVALID EVENT DECLARATION
275       MUST BE DIRECT FILE NAME
276       VALUE CLAUSE NOT ALLOWED FOR GLOBAL OR OWN ITEM
277       INTO EXPECTED
```

278 FROM EXPECTED
279 DIRECT I-O RECORD AREA REQUIRED
280 INVALID CONTROL VARIABLE FOR PERFORM VARYING
281 INVALID PROGRAM SPECIFICATION
282 EVENT NAME OPERAND EXPECTED
283 CANT PERFORM DECLARATIVE THAT USES 'FILE' OPTION
284 INVALID PARAMETER
285 ARRAY ROW MAY NOT BE DEALLOCATED
286 DICTIONARY OVERFLOW - PROGRAM TOO LARGE
287 SIGN CLAUSE MAY ONLY APPLY TO DISPLAY ITEM
288 NULL OPTION NOT ALLOWED FOR THIS ITEM TYPE
289 DUPLICATE PROCEDURE DIVISION USING PARAMETER
290 TASK ATTRIBUTE EXPECTED
291 DIRECT I-O BUFFER ATTRIBUTE EXPECTED
292 ILLEGAL ACCESS MODE
293 USER OPTION MUST BE ALPHANUMERIC IDENTIFIER
294 LOCKABLE OPERAND EXPECTED
295 INVALID FROM OR THRU CLAUSE IN COPY STATEMENT
296 NESTED COPY STATEMENTS ARE NOT ALLOWED
297 INVALID OPERAND FOR CALL SYSTEM WITH
298 RECORD CONTAINS CLAUSE IGNORED
299 BLOCK SIZE MUST BE STATED IN CHARACTERS OR WORDS
300 MAXIMUM RECORD SIZE > BLOCK SIZE
301 MOVE TRUNCATION
302 POSSIBLE TRUNCATION OF NON-ZERO DIGITS
303 MISSING END DECLARATIVES
304 MISSING RECORD DESCRIPTION
305 MUST BE DISPLAY OR DISPLAY-1
306 GROUP COMPUTATIONAL ITEMS CAN NOT BE DISPLAYED
307 DATA-BASE OPEN ERROR
308 INVALID SELECTION EXPRESSION
309 INVALID DATA BASE EXCEPTION EXPRESSION
310 SET NOT IN DATA BASE DIRECTORY
311 INVOKED SET IN ERROR
312 MUST BE PROCEDURE NAME
313 DOLLAR PARAMETER DOES NOT HAVE BOOLEAN VALUE
314 MUST BE WORD OR INTEGER
315 FILE HAS NO LINAGE CLAUSE
316 SET NAME EXPECTED
317 INVALID DATA BASE SET STATEMENT
318 DATA BASE OPERAND EXPECTED
319 BLOCK SIZE MAY BE TOO SMALL
320 FILES INTERNAL MODE CONFLICTS WITH SORT FILE
321 POSSIBLE SHORT BLOCK PROBLEM
322 ILLEGAL USE OF LOCK VERB
323 MISSING FILE DESCRIPTION
324 MISSING PROGRAM
325 BEFORE ACTION TAKEN FOR DIRECT FILES
326 ILLEGAL HARDWARE TYPE FOR DIRECT FILES
327 BLOCK OR REPORTS CLAUSE ILLEGAL FOR DIRECT FILE
328 MUST BE LD NAME
329 MUST BE WITHIN CURRENT LD DESCRIPTION
330 INTERRUPT PROCEDURE EXPECTED
331 REPORT FILES MUST BE ASSIGNED TO PRINTER
332 ARITHMETIC FUNCTION REQUIRED
333 FORMAL PARAMETER ERROR
334 ACTUAL PARAMETER ERROR
335 SEQUENCE ERROR ON SOURCE OUTPUT FILE
336 MINIMUM RECORD SIZE IGNORED

337 ILLEGAL USE OF LOWER-BOUNDS CLAUSE
338 STOP RUN WILL EXIT THE GLOBAL BLOCK
339 NON-EXECUTABLE STATEMENT
340 ONLY LEVEL 1 ITEMS MAY BE RECEIVED AS ARRAYS
341 ARRAYS ARE PASSED BY NAME (REFERENCE)
342 ILLEGAL USE OF SEGMENT CLAUSE
343 STOP RUN IS ASSUMED
344 EXIT PROGRAM WILL CONTINUE BACK TO CO-ROUTINE
345 AREASIZE NOT A MULTIPLE OF BLOCKING RATIO
346 RECORD DESCRIPTIONS NOT ALLOWED FOR REPORT FILES
347 NO RD ENTRY GIVEN FOR THIS REPORT
348 LABELS AND DATA-NAMES MUST NOT BE RESERVED WORDS
349 GLOBAL OR OWN CAN NOT BE DECLARED AT LEVEL 2
350 FAMILY ATTRIBUTE EXPECTED
351 ILLEGAL USE OF GLOBAL OR OWN CLAUSE
352 FILE ALREADY NAMED IN SAME RECORD AREA CLAUSE
353 USE QUOTED LITERAL IF EDITING IS NOT DESIRED
354 DM ATTRIBUTE EXPECTED
355 GROUP ITEM REQUIRED
356 STRING OR FIGURATIVE MUST FOLLOW ALL
357 MAXIMUM ITEM SIZE EXCEEDED
358 OCCURS GREATER THAN 65535 ARE NOT ALLOWED
359 INVALID CHARACTER IN UNDIGIT LITERAL
360 ILLEGAL USAGE FOR MONITOR/DUMP FILE
361 INVALID LEXICOGRAPHICAL LEVEL
362 OPTION IGNORED AFTER IDENTIFICATION DIVISION
363 INFO TABLE OVERFLOW, TOO MANY DATA-NAMES USED
364 SD FILE -- ILLEGAL ASCII INTMODE
365 DECLARED AS RECEIVED BUT NOT IN PARAMETER LIST
366 $FROM NOT ALLOWED IN LIBRARY OR SAVED INPUT
367 ONLY INITIALIZE ALLOWED FOR PARTITION
368 GLOBAL OR OWN DATA-NAMES CANNOT BE PARAMETERS
369 ON EXCEPTION EXPECTED
370 ONLY ONE DATA-BASE-RESTART FILE IS ALLOWED
371 NO DATA-BASE-RESTART FILE DECLARED
372 DIRECT FILES NOT ALLOWED FOR RESTART FILES
373 MUST BE DATA-BASE-RESTART FILE
374 ALL REPORTS ON A FILE MUST HAVE THE SAME USAGE
375 ILLEGAL CLAUSE USED WITH DIRECT INDEX FILE
376 NUMERIC OR EDITED NUMERIC ASCII IS NOT ALLOWED
377 GLOBAL OR RECEIVED BY REFERENCE CLAUSE EXPECTED
378 FLOATING POINT ITEM MAY NOT HAVE PICTURE CLAUSE
379 COMPILE AND GO NOT LEGAL WITH PARAMETERS
380 DM ERROR
381 NAME EXCEEDS 17 CHARACTERS
382 SECURITY VIOLATION
383 VERSION ERROR
384 DATA BASE DOES NOT EXIST
385 DATA SET NAME EXPECTED
386 DUPLICATE DMERROR USE PROCEDURE
387 DM COUNT OR RECORD TYPE EXPECTED
388 DUPLICATE EXPECTED
389 INVALID DM KEY CONDITION
390 LINK EXPECTED
391 SET NAME EXPECTED
392 MUST BE MANUAL SET
393 SEGMENT DICTIONARY SIZE EXCEEDED
394 GLOBAL STACK SIZE EXCEEDED
395 STACK SIZE EXCEEDED

396	PARAMETER WILL BE PASSED BY REFERENCE
397	SYNCHRONIZED CLAUSE IGNORED
398	MUST COMPILE COBOL COMPILER WITH BDMS OPTION SET
399	IDENTIFICATION DIVISION MISSING OR MIS-SPELLED
400	ENVIRONMENT DIVISION MISSING OR MIS-SPELLED
401	MUST BE AN IMPERATIVE STATEMENT
402	COBOL TO DATABASE/INTERFACE VERSION MIS-MATCH
403	SUBSCRIPT WILL CAUSE INVALID INDEX AT RUN TIME
404	FILE SIZE IGNORED FOR SORT DISK
405	ATTRIBUTE WILL CAUSE NON-FATAL RUN-TIME ERROR
406	MULTIPLE RECEIVING FIELDS ILLEGAL WITH REMAINDER
407	CHARACTER MAY NOT BE USED MORE THAN ONCE
408	ALPHABET-NAME IN COLLATING SEQ CLAUSE MISSING
409	NUMERIC LITERAL MUST BE IN RANGE OF 1 TO 256
410	LITERAL MUST BE SINGLE CHARACTER
411	MAY ONLY BE USED WITH INDEXED FILE
412	MAY NOT BE USED WITH INDEXED FILE
413	CLOSE OPTION NOT MEANINGFUL FOR FILE KIND
414	MUST BE NATIVE, STANDARD-1, OR BCL CODE-SET
415	CONSTRUCT ALLOWED ONLY UNDER ANSI74 $ OPTION
416	CONSTRUCT NOT ALLOWED UNDER ANSI74 $ OPTION
417	FILE DECLARATION MUST CONTAIN ASSIGN CLAUSE
418	ILLEGAL HARDWARE TYPE FOR ACTUAL KEY CLAUSE
419	CONSTRUCT ALLOWED ONLY UNDER B2500 $ OPTION

WARNING MESSAGES PRODUCED BY $ ANALYZE

ALTER STATEMENTS SHOULD BE AVOIDED

An ALTER statement may complicate debugging and maintenance functions.

DISPLAYS AND ACCEPTS SHOULD BE AVOIDED

These statements produce large amounts of code and, because of the slow speed devices which these statements access, tend to degrade execution time.

J-SIGN PICTURES - AVOID WHERE POSSIBLE

PICTURE clauses which specify the PICTURE character "J" cause considerable amounts of extra code to be generated and execution time to be degraded.

LABELS PERFORMED OR GONE TO FROM DIFFERENT SEGMENT

This may cause repeated presence bit action on code segments.

NON-CONSTANT VALUE USED AS SUBSCRIPT

When an identifier is used as a subscript, evaluation of the subscript is done at run time.

ON SIZE ERROR CLAUSE

The presence of the ON SIZE ERROR clause will cause extra code to be generated and executed. Proper planning of field sizes will eliminate the need for this clause.

PHYSICAL BLOCKSIZE NOT MULTIPLE OF 30 WORDS

The size of a physical block for this disk file is not a multiple of 180 EBCDIC (240 BCL) characters and thus will waste space on disk.

SORTS WITH BCL KEYS

A SORT of a BCL file requires conversion of the keys to EBCDIC because of differences in collating sequence.

SORTS WITH MULTIPLE KEYS

Multiple keys require that the compiler generate a compare procedure instead of allowing the in-line compare within the SORT to be used.

STATEMENT CAUSED CONVERSION FROM BINARY TO DECIMAL

The statement flagged will cause a conversion from binary so that data may be stored in a DISPLAY field.

STATEMENT CAUSED CONVERSION FROM DECIMAL TO BINARY

The statement flagged will cause a conversion from DISPLAY so that data may be stored in a binary field.

G. CHARACTER REPRESENTATION, COLLATING SEQUENCE AND TRANSLATION

EBCDIC Collating Sequence

SYMBOL	REPRESENTATION BINARY		HEX	CARD CODE	
NUL	0000	0000	00	12-0-9-8-1	LOW
SOH	0000	0001	01	12-9-1	
STX	0000	0010	02	12-9-2	
ETX	0000	0011	03	12-9-3	
	0000	0100	04	12-9-4	
HT	0000	0101	05	12-9-5	
	0000	0110	06	12-9-6	
DEL	0000	0111	07	12-9-7	
	0000	1000	08	12-9-8	
	0000	1001	09	12-9-8-1	
	0000	1010	0A	12-9-8-2	
VT	0000	1011	0B	12-9-8-3	
FF	0000	1100	0C	12-9-8-4	
CR	0000	1101	0D	12-9-8-5	
SO	0000	1110	0E	12-9-8-6	
ST	0000	1111	0F	12-9-8-7	
DLE	0001	0000	10	12-11-9-8-1	
DC1	0001	0001	11	11-9-1	
DC2	0001	0010	12	11-9-2	
DC3	0001	0011	13	11-9-3	
	0001	0100	14	11-9-4	
NL	0001	0101	15	11-9-5	
BS	0001	0110	16	11-9-6	
	0001	0111	17	11-9-7	
CAN	0001	1000	18	11-9-8	
EM	0001	1001	19	11-9-8-1	
	0001	1010	1A	11-9-8-2	
	0001	1011	1B	11-9-8-3	
FS	0001	1100	1C	11-9-8-4	
GS	0001	1101	1D	11-9-8-5	
RS	0001	1110	1E	11-9-8-6	
US	0001	1111	1F	11-9-8-7	
	0010	0000	20	11-0-9-8-1	
	0010	0001	21	0-9-1	
	0010	0010	22	0-9-2	
	0010	0011	23	0-9-3	
	0010	0100	24	0-9-4	
LF	0010	0101	25	0-9-5	
ETB	0010	0110	26	0-9-6	
ESC	0010	0111	27	0-9-7	
	0010	1000	28	0-9-8	
	0010	1001	29	0-9-8-1	
	0010	1010	2A	0-9-8-2	
	0010	1011	2B	0-9-8-3	
	0010	1100	2C	0-9-8-4	
ENQ	0010	1101	2D	0-9-8-5	
ACK	0010	1110	2E	0-9-8-6	
BEL	0010	1111	2F	0-9-8-7	
	0011	0000	30	12-11-0-9-8-1	
	0011	0001	31	9-1	
SYN	0011	0010	32	9-2	
	0011	0011	33	9-3	
	0011	0100	34	9-4	
	0011	0101	35	9-5	
	0011	0110	36	9-6	
EOT	0011	0111	37	9-7	
	0011	1000	38	9-8	
	0011	1001	39	9-8-1	
	0011	1010	3A	9-8-2	
	0011	1011	3B	9-8-3	
DC4	0011	1100	3C	9-8-4	
NAK	0011	1101	3D	9-8-5	
	0011	1110	3E	9-8-6	
SUB	0011	1111	3F	9-8-7	HIGH

EBCDIC Collating Sequence (cont)

SYMBOL	REPRESENTATION BINARY	HEX	CARD CODE		
SP	0100 0000	40			Blank Space
	0100 0001	41	12-0-9-1	LOW	
	0100 0010	42	12-0-9-2		
	0100 0011	43	12-0-9-3		
	0100 0100	44	12-0-9-4		
	0100 0101	45	12-0-9-5		
	0100 0110	46	12-0-9-6		
	0100 0111	47	12-0-9-7		
	0100 1000	48	12-0-9-8		
	0100 1001	49	12-8-1		
[	0100 1010	4A	12-8-2		
.	0100 1011	4B	12-8-3		
<	0100 1100	4C	12-8-4		
(	0100 1101	4D	12-8-5		
+	0100 1110	4E	12-8-6		
!	0100 1111	4F	12-8-7		
&	0101 0000	50	12		
	0101 0001	51	12-11-9-1		
	0101 0010	52	12-11-9-2		
	0101 0011	53	12-11-9-3		
	0101 0100	54	12-11-9-4		
	0101 0101	55	12-11-9-5		
	0101 0110	56	12-11-9-6		
	0101 0111	57	12-11-9-7		
	0101 1000	58	12-11-9-8		
	0101 1001	59	11-8-1		
]	0101 1010	5A	11-8-2		
$	0101 1011	5B	11-8-3		
*	0101 1100	5C	11-8-4		
)	0101 1101	5D	11-8-5		
;	0101 1110	5E	11-8-6		
¬	0101 1111	5F	11-8-7		Logical Not
-	0110 0000	60	11		
/	0110 0001	61	0-1		
	0110 0010	62	11-0-9-2		
	0110 0011	63	11-0-9-3		
	0110 0100	64	11-0-9-4		
	0110 0101	65	11-0-9-5		
	0110 0110	66	11-0-9-6		
	0110 0111	67	11-0-9-7		
	0110 1000	68	11-0-9-8		
	0110 1001	69	0-8-1		
	0110 1010	6A	12-11		
,	0110 1011	6B	0-8-3		
%	0110 1100	6C	0-8-4		
_	0110 1101	6D	0-8-5		Underscore
>	0110 1110	6E	0-8-6		
?	0110 1111	6F	0-8-7		
	0111 0000	70	12-11-0		
	0111 0001	71	12-11-0-9-1		
	0111 0010	72	12-11-0-9-2		
	0111 0011	73	12-11-0-9-3		
	0111 0100	74	12-11-0-9-4		
	0111 0101	75	12-11-0-9-5		
	0111 0110	76	12-11-0-9-6		
	0111 0111	77	12-11-0-9-7		
	0111 1000	78	12-11-0-9-8		
	0111 1001	79	8-1		
:	0111 1010	7A	8-2		
#	0111 1011	7B	8-3		
@	0111 1100	7C	8-4		
'	0111 1101	7D	8-5	HIGH	Apostrophe
=	0111 1110	7E	8-6		
"	0111 1111	7F	8-7		

EBCDIC Collating Sequence (cont)

SYMBOL	REPRESENTATION BINARY	HEX	CARD CODE		
	1000 0000	80	12-0-8-1	LOW	
a	1000 0001	81	12-0-1		Lower Case A
b	1000 0010	82	12-0-2		Lower Case B
c	1000 0011	83	12-0-3		Lower Case C
d	1000 0100	84	12-0-4		Lower Case D
e	1000 0101	85	12-0-5		Lower Case E
f	1000 0110	86	12-0-6		Lower Case F
g	1000 0111	87	12-0-7		Lower Case G
h	1000 1000	88	12-0-8		Lower Case H
i	1000 1001	89	12-0-9		Lower Case I
	1000 1010	8A	12-0-8-2		
	1000 1011	8B	12-0-8-3		
	1000 1100	8C	12-0-8-4		
	1000 1101	8D	12-0-8-5		
	1000 1110	8E	12-0-8-6		
	1000 1111	8F	12-0-8-7		
	1001 0000	90	12-11-8-1		
j	1001 0001	91	12-11-1		Lower Case J
k	1001 0010	92	12-11-2		Lower Case K
l	1001 0011	93	12-11-3		Lower Case L
m	1001 0100	94	12-11-4		Lower Case M
n	1001 0101	95	12-11-5		Lower Case N
o	1001 0110	96	12-11-6		Lower Case O
p	1001 0111	97	12-11-7		Lower Case P
q	1001 1000	98	12-11-8		Lower Case Q
r	1001 1001	99	12-11-9		Lower Case R
	1001 1010	9A	12-11-8-2		
	1001 1011	9B	12-11-8-3		
	1001 1100	9C	12-11-8-4		
	1001 1101	9D	12-11-8-5		
	1001 1110	9E	12-11-8-6		
	1001 1111	9F	12-11-8-7		
	1010 0000	A0	11-0-8-1		
	1010 0001	A1	11-0-1		
s	1010 0010	A2	11-0-2		Lower Case S
t	1010 0011	A3	11-0-3		Lower Case T
u	1010 0100	A4	11-0-4		Lower Case U
v	1010 0101	A5	11-0-5		Lower Case V
w	1010 0110	A6	11-0-6		Lower Case W
x	1010 0111	A7	11-0-7		Lower Case X
y	1010 1000	A8	11-0-8		Lower Case Y
z	1010 1001	A9	11-0-9		Lower Case Z
	1010 1010	AA	11-0-8-2		
	1010 1011	AB	11-0-8-3		
	1010 1100	AC	11-0-8-4		
	1010 1101	AD	11-0-8-5		
	1010 1110	AE	11-0-8-6		
	1010 1111	AF	11-0-8-7		
	1011 0000	B0	12-11-0-8-1		
	1011 0001	B1	12-11-0-1		
	1011 0010	B2	12-11-0-2		
	1011 0011	B3	12-11-0-3		
	1011 0100	B4	12-11-0-4		
	1011 0101	B5	12-11-0-5		
	1011 0110	B6	12-11-0-6		
	1011 0111	B7	12-11-0-7		
	1011 1000	B8	12-11-0-8		
	1011 1001	B9	12-11-0-9		
	1011 1010	BA	12-11-0-8-2		
	1011 1011	BB	12-11-0-8-3		
	1011 1100	BC	12-11-0-8-4		
	1011 1101	BD	12-11-0-8-5		
	1011 1110	BE	12-11-0-8-6		
	1011 1111	BF	12-11-0-8-7	HIGH	

EBCDIC Collating Sequence (cont)

SYMBOL	REPRESENTATION BINARY	HEX	CARD CODE		
+	1100 0000	C0	12-0	LOW	
A	1100 0001	C1	12-1		
B	1100 0010	C2	12-2		
C	1100 0011	C3	12-3		
D	1100 0100	C4	12-4		
E	1100 0101	C5	12-5		
F	1100 0110	C6	12-6		
G	1100 0111	C7	12-7		
H	1100 1000	C8	12-8		
I	1100 1001	C9	12-9		
	1100 1010	CA	12-0-9-8-2		
	1100 1011	CB	12-0-9-8-3		
	1100 1100	CC	12-0-9-8-4		
	1100 1101	CD	12-0-9-8-5		
	1100 1110	CE	12-0-9-8-6		
	1100 1111	CF	12-0-9-8-7		
-	1101 0000	D0	11-0		
J	1101 0001	D1	11-1		
K	1101 0010	D2	11-2		
L	1101 0011	D3	11-3		
M	1101 0100	D4	11-4		
N	1101 0101	D5	11-5		
O	1101 0110	D6	11-6		
P	1101 0111	D7	11-7		
Q	1101 1000	D8	11-8		
R	1101 1001	D9	11-9		
	1101 1010	DA	12-11-9-8-2		
	1101 1011	DB	12-11-9-8-3		
	1101 1100	DC	12-11-9-8-4		
	1101 1101	DD	12-11-9-8-5		
	1101 1110	DE	12-11-9-8-6		
	1101 1111	DF	12-11-9-8-7		
	1110 0000	E0	0-8-2		Back Slash
\	1110 0001	E1	11-0-9-1		
S	1110 0010	E2	0-2		
T	1110 0011	E3	0-3		
U	1110 0100	E4	0-4		
V	1110 0101	E5	0-5		
W	1110 0110	E6	0-6		
X	1110 0111	E7	0-7		
Y	1110 1000	E8	0-8		
Z	1110 1001	E9	0-9		
	1110 1010	EA	11-0-9-8-2		
	1110 1011	EB	11-0-9-8-3		
	1110 1100	EC	11-0-9-8-4		
	1110 1101	ED	11-0-9-8-5		
	1110 1110	EF	11-0-9-8-6		
	1110 1111	EF	11-0-9-8-7		
0	1111 0000	F0	0		
1	1111 0001	F1	1		
2	1111 0010	F2	2		
3	1111 0011	F3	3		
4	1111 0100	F4	4		
5	1111 0101	F5	5		
6	1111 0110	F6	6		
7	1111 0111	F7	7		
8	1111 1000	F8	8		
9	1111 1001	F9	9		
	1111 1010	FA	12-11-0-9-8-2		
	1111 1011	FB	12-11-0-9-8-3		
	1111 1100	FC	12-11-0-9-8-4		
	1111 1101	FD	12-11-0-9-8-5		
	1111 1110	FE	12-11-0-9-8-6		
	1111 1111	FF	12-11-0-9-8-7	HIGH	

BCL to EBCDIC Translation Chart

EBCDIC Collating Sequence Order

BCL CHARACTER	BCL EXTERNAL BA	4321	BCL INTERNAL BA	8421	BCL CARD CODE		TRANSLATED EBCDIC CODE	EBCDIC HEX	EBCDIC CARD CODE	
(Blank)	01	0000	11	0000	-	-	0100 0000	40	-	-
[	11	1100	01	1011	12	8-4	0100 1010	4A	12	8-2
.	11	1011	01	1010	12	8-3	0100 1011	4B		8-3
<	11	1110	01	1110	12	8-6	0100 1100	4C		8-4
(	11	1101	01	1101	12	8-5	0100 1101	4D		8-5
+	11	1111	01	1111	12	8-7	0100 1111	4F		8-7
&	11	0000	01	1100	12	-	0101 0000	50	12	-
]	01	1110	11	1110	0	8-6	0101 1010	5A	11	8-2
$	10	1011	10	1010	11	8-3	0101 1011	5B	11	8-3
*	10	1100	10	1011	11	8-4	0101 1100	5C	11	8-4
)	10	1101	10	1101	11	8-5	0101 1101	5D	11	8-5
;	10	1110	10	1110	11	8-6	0101 1110	5E	11	8-6
≤	10	1111	10	1111	11	8-7	0101 1111	5F	11	8-7
-	10	0000	10	1100	11	-	0110 0000	60	11	-
/	01	0001	10	0001	0	1	0110 0001	61	0	1
,	01	1011	11	1010	0	8-3	0110 1011	6B	0	8-3
%	01	1100	11	1011	0	8-4	0110 1100	6C	0	8-4
≠	01	1010	11	1100	0	8-2	0110 1101	6D	0	8-5
>	00	1110	00	1110	-	8-6	0110 1110	6E	0	8-6
?	00	0000	00	1100	All other card codes		0110 1111	6F	0	8-7
:	00	1101	00	1101	-	8-5	0111 1010	7A	-	8-2
#	00	1011	00	1010	-	8-3	0111 1011	7B	-	8-3
@	00	1100	00	1011	-	8-4	0111 1100	7C	-	8-4
≥	00	1111	00	1111	-	8-7	0111 1101	7D	-	8-5
=	01	1101	11	1101	0	8-5	0111 1110	7E	-	8-6
"	01	1111	11	1111	0	8-7	0111 1111	7F	-	8-7
+	11	1010	01	0000	12	0	1100 0000	C0	12	0
A	11	0001	01	0001	12	1	1100 0001	C1	12	1
B	11	0010	01	0010	12	2	1100 0010	C2	12	2
C	11	0011	01	0011	12	3	1100 0011	C3	12	3
D	11	0100	01	0100	12	4	1100 0100	C4	12	4
E	11	0101	01	0101	12	5	1100 0101	C5	12	5
F	11	0110	01	0110	12	6	1100 0110	C6	12	6
G	11	0111	01	0111	12	7	1100 0111	C7	12	7
H	11	1000	01	1000	12	8	1100 1000	C8	12	8
I	11	1001	01	1001	12	9	1100 1001	C9	12	9
X	10	1010	10	0000	11	0	1101 0000	D0	11	0
J	10	0001	10	0001	11	1	1101 0001	D1	11	1
K	10	0010	10	0010	11	2	1101 0010	D2	11	2
L	10	0011	10	0011	11	3	1101 0011	D3	11	3
M	10	0100	10	0100	11	4	1101 0100	D4	11	4
N	10	0101	10	0101	11	5	1101 0101	D5	11	5
O	10	0110	10	0110	11	6	1101 0110	D6	11	6
P	10	0111	10	0111	11	7	1101 0111	D7	11	7
Q	10	1000	10	1000	11	8	1101 1000	D8	11	8
R	10	1001	10	1001	11	9	1101 1001	D9	11	9
S	01	0010	11	0010	0	2	1110 0010	E2	0	2
T	01	0011	11	0011	0	3	1110 0011	E3	0	3
U	01	0100	11	0100	0	4	1110 0100	E4	0	4
V	01	0101	11	0101	0	5	1110 0101	E5	0	5
W	01	0110	11	0110	0	6	1110 0110	E6	0	6
X	01	0111	11	0111	0	7	1110 0111	E7	0	7
Y	01	1000	11	1000	0	8	1110 1000	E8	0	8
Z	01	1001	11	1001	0	9	1110 1001	E9	0	9
0	00	1010	00	0000	-	0	1111 0000	F0	-	0
1	00	0001	00	0001	-	1	1111 0001	F1	-	1
2	00	0010	00	0010	-	2	1111 0010	F2	-	2
3	00	0011	00	0011	-	3	1111 0011	F3	-	3
4	00	0100	00	0100	-	4	1111 0100	F4	-	4
5	00	0101	00	0101	-	5	1111 0101	F5	-	5
6	00	0110	00	0110	-	6	1111 0110	F6	-	6
7	00	0111	00	0111	-	7	1111 0111	F7	-	7
8	00	1000	00	1000	-	8	1111 1000	F8	-	8
9	00	1001	00	1001	-	9	1111 1001	F9	-	9

LOW ↑ ↓ HIGH

ASCII to EBCDIC Translation Chart

ASCII Collating Sequence Order

SYMBOL	ASCII BINARY		ASCII HEX	EBCDIC BINARY		EBCDIC HEX	CARD CODE	
NUL	0000	0000	00	0000	0000	00	12-0-9-8-1	
SOH	0000	0001	01	0000	0001	01	12-9-1	
STX	0000	0010	02	0000	0010	02	12-9-2	
FTX	0000	0011	03	0000	0011	03	12-9-3	
EOT	0000	0100	04	0011	0111	37	9-7	
ENQ	0000	0101	05	0010	1101	2D	0-9-8-5	
CCK	0000	0110	06	0010	1110	2F	0-9-8-6	
BEL	0000	0111	07	0010	1111	2F	0-9-8-7	
BS	0000	1000	08	0001	0110	16	11-9-6	
HT	0000	1001	09	0000	0101	05	12-9-5	
LF	0000	1010	0A	0010	0101	25	0-9-5	
VT	0000	1011	0B	0000	1011	0B	12-9-8-3	
FF	0000	1100	0C	0000	1100	0C	12-9-8-4	
CR	0000	1101	0D	0000	1101	0D	12-9-8-5	
SO	0000	1110	0E	0000	1110	0F	12-9-8-6	
SI	0000	1111	0F	0000	1111	0F	12-9-8-7	
DLE	0001	0000	10	0001	0000	10	12-11-9-8-1	
DC1	0001	0001	11	0001	0001	11	11-9-1	
DC2	0001	0010	12	0001	0010	12	11-9-2	
DC3	0001	0011	13	0001	0011	13	11-9-3	
DC4	0001	0100	14	0011	1100	3C	9-8-4	
NAK	0001	0101	15	0011	1101	3D	9-8-5	
SYN	0001	0110	16	0011	0010	32	9-2	
ETR	0001	0111	17	0010	0110	26	0-9-6	
CAN	0001	1000	18	0001	1000	18	11-9-8	
EM	0001	1001	19	0001	1001	19	11-9-8-1	
SUB	0001	1010	1A	0011	1111	3F	9-8-7	
ESC	0001	1011	1B	0010	0111	27	0-9-7	
FS	0001	1100	1C	0001	1100	1C	11-9-8-4	
GS	0001	1101	1D	0001	1101	1D	11-9-8-5	
RS	0001	1110	1E	0001	1110	1E	11-9-8-6	
US	0001	1111	1F	0001	1111	1F	11-9-8-7	
SP	0010	0000	20	0100	0000	40		Blank Space
!	0010	0001	21	0100	1111	4F	12-8-7	Exclamation Point (left arrow)
"	0010	0010	22	0111	1111	7F	8-7	
#	0010	0011	23	0111	1011	7B	8-3	
$	0010	0100	24	0101	1011	5B	11-8-3	
%	0010	0101	25	0110	1100	6C	0-8-4	
&	0010	0110	26	0101	0000	50	12	
'	0010	0111	27	0111	1101	7D	8-5	
(	0010	1000	28	0100	1101	4D	12-8-5	
)	0010	1001	29	0101	1101	5D	11-8-5	
*	0010	1010	2A	0101	1100	5C	11-8-4	
+	0010	1011	2B	0100	1110	4E	12-8-6	
,	0010	1100	2C	0110	1011	6B	0-8-3	
-	0010	1101	2D	0110	0000	60	11	
.	0010	1110	2E	0100	1011	4B	12-8-3	
/	0010	1111	2F	0110	0001	61	0-1	
0	0011	0000	30	1111	0000	F0	0	
1	0011	0001	31	1111	0001	F1	1	
2	0011	0010	32	1111	0010	F2	2	
3	0011	0011	33	1111	0011	F3	3	
4	0011	0100	34	1111	0100	F4	4	
5	0011	0101	35	1111	0101	F5	5	
6	0011	0110	36	1111	0110	F6	6	
7	0011	0111	37	1111	0111	F7	7	
8	0011	1000	38	1111	1000	F8	8	
9	0011	1001	39	1111	1001	F9	9	
:	0011	1010	3A	0111	1010	7A	8-2	
;	0011	1011	3B	0101	1110	5E	11-8-6	
<	0011	1100	3C	0100	1100	4C	12-8-4	
=	0011	1101	3D	0111	1110	7E	8-6	
>	0011	1110	3E	0110	1110	6E	0-8-6	
	0011	1111	3F	0110	1111	6F	0-8-7	

ASCII to EBCDIC Translation Chart

ASCII Collating Sequence Order (cont)

SYMBOL	ASCII BINARY	HEX	EBCDIC BINARY	HEX	CARD CODE	
@	0100 0000	40	0111 1100	7C	8-4	
A	0100 0001	41	1100 0001	C1	12-1	
B	0100 0010	42	1100 0010	C2	12-2	
C	0100 0011	43	1100 0011	C3	12-3	
D	0100 0100	44	1100 0100	C4	12-4	
E	0100 0101	45	1100 0101	C5	12-5	
F	0100 0110	46	1100 0110	C6	12-6	
G	0100 0111	47	1100 0111	C7	12-7	
H	0100 1000	48	1100 1000	C8	12-8	
I	0100 1001	49	1100 1001	C9	12-9	
J	0100 1010	4A	1101 0001	D1	11-1	
K	0100 1011	4B	1101 0010	D2	11-2	
L	0100 1100	4C	1101 0011	D3	11-3	
M	0100 1101	4D	1101 0100	D4	11-4	
N	0100 1110	4E	1101 0101	D5	11-5	
O	0100 1111	4F	1101 0110	D6	11-6	
P	0101 0000	50	1101 0111	D7	11-7	
Q	0101 0001	51	1101 1000	D8	11-8	
R	0101 0010	52	1101 1001	D9	11-9	
S	0101 0011	53	1110 0010	E2	0-2	
T	0101 0100	54	1110 0011	E3	0-3	
U	0101 0101	55	1110 0100	E4	0-4	
V	0101 0110	56	1110 0101	E5	0-5	
W	0101 0111	57	1110 0110	E6	0-6	
X	0101 1000	58	1110 0111	E7	0-7	
Y	0101 1001	59	1110 1000	E8	0-8	
Z	0101 1010	5A	1110 1001	E9	0-9	
[	0101 1011	5B	0100 1010	4A	12-8-2	
\	0101 1100	5C	1110 0000	E0	0-8-2	Back Slash
]	0101 1101	5D	0101 1010	5A	11-8-2	
¬	0101 1110	5E	0101 1111	5F	11-8-7	Logical Not (LEQ Symbol)
—	0101 1111	5F	0110 1101	6D	0-8-5	Underscore (Unequal Symbol)
	0110 0000	60	0111 1001	79	8-1	Grave Accent
a	0110 0001	61	1000 0001	81	12-0-1	Lower Case A
b	0110 0010	62	1000 0010	82	12-0-2	Lower Case B
c	0110 0011	63	1000 0011	83	12-0-3	Lower Case C
d	0110 0100	64	1000 0100	84	12-0-4	Lower Case D
e	0110 0101	65	1000 0101	85	12-0-5	Lower Case E
f	0110 0110	66	1000 0110	86	12-0-6	Lower Case F
g	0110 0111	67	1000 0111	87	12-0-7	Lower Case G
h	0110 1000	68	1000 1000	88	12-0-8	Lower Case H
i	0110 1001	69	1000 1001	89	12-0-9	Lower Case I
j	0110 1010	6A	1001 0001	91	12-11-1	Lower Case J
k	0110 1011	6B	1001 0010	92	12-11-2	Lower Case K
l	0110 1100	6C	1001 0011	93	12-11-3	Lower Case L
m	0110 1101	6D	1001 0100	94	12-11-4	Lower Case M
n	0110 1110	6E	1001 0101	95	12-11-5	Lower Case N
o	0110 1111	6F	1001 0110	96	12-11-6	Lower Case O
p	0111 0000	70	1001 0111	97	12-11-7	Lower Case P
q	0111 0001	71	1001 1000	98	12-11-8	Lower Case Q
r	0111 0010	72	1001 1001	99	12-11-9	Lower Case R
s	0111 0011	73	1010 0010	A2	11-0-2	Lower Case S
t	0111 0100	74	1010 0011	A3	11-0-3	Lower Case T
u	0111 0101	75	1010 0100	A4	11-0-4	Lower Case U
v	0111 0110	76	1010 0101	A5	11-0-5	Lower Case V
w	0111 0111	77	1010 0110	A6	11-0-6	Lower Case W
x	0111 1000	78	1010 0111	A7	11-0-7	Lower Case X
y	0111 1001	79	1010 1000	A8	11-0-8	Lower Case Y
z	0111 1010	7A	1010 1001	A9	11-0-9	Lower Case Z
	0111 1011	7B	1100 0000	C0	12-0	Left Brace
	0111 1100	7C	0110 1010	6A	12-11	Broken Vertical Line
	0111 1101	7D	1101 0000	D0	11-0	Right Brace
	0111 1110	7E	1010 0001	A1	11-0-1	Tilde
DEL	0111 1111	7F	0000 0111	D7	12-9-7	

ASCII to EBCDIC Translation Chart

ASCII Collating Sequence Order (cont)

SYMBOL	ASCII BINARY	ASCII HEX	EBCDIC BINARY	EBCDIC HEX	CARD CODE
K0	1000 0000	80	0010 0000	20	11-0-9-8-1
K1	1000 0001	81	0010 0001	21	0-9-1
K2	1000 0010	82	0010 0010	22	0-9-2
K3	1000 0011	83	0010 0011	23	0-9-3
K4	1000 0100	84	0010 0100	24	0-9-4
K5	1000 0101	85	0001 0101	15	11-9-5
K6	1000 0110	86	0000 0110	06	12-9-6
K7	1000 0111	87	0001 0111	17	11-9-7
K8	1000 1000	88	0010 1000	28	0-9-8
K9	1000 1001	89	0010 1001	29	0-9-8-1
K10	1000 1010	8A	0010 1010	2A	0-9-8-2
K11	1000 1011	8B	0010 1011	2B	0-9-8-3
K12	1000 1100	8C	0010 1100	2C	0-9-8-4
K13	1000 1101	8D	0000 1001	09	12-9-8-1
K14	1000 1110	8E	0000 1010	0A	12-9-8-2
K15	1000 1111	8F	0001 1011	1B	11-9-8-3
K16	1001 0000	90	0011 0000	30	12-11-0-9-8-1
K17	1001 0001	91	0011 0001	31	9-1
K18	1001 0010	92	0001 1010	1A	11-9-8-2
K19	1001 0011	93	0011 0011	33	9-3
K20	1001 0100	94	0011 0100	34	9-4
K21	1001 0101	95	0011 0101	35	9-5
K22	1001 0110	96	0011 0110	36	9-6
K23	1001 0111	97	0000 1000	08	12-9-8
K24	1001 1000	98	0011 1000	38	9-8
K25	1001 1001	99	0011 1001	39	9-8-1
K26	1001 1010	9A	0011 1010	3A	9-8-2
K27	1001 1011	9B	0011 1011	3B	9-8-3
K28	1001 1100	9C	0000 0100	04	12-9-4
K29	1001 1101	9D	0001 0100	14	11-9-4
K30	1001 1110	9E	0011 1110	3E	9-8-6
K31	1001 1111	9F	1110 0001	E1	11-0-9-1
N0	1010 0000	A0	0100 0001	41	12-0-9-1
N1	1010 0001	A1	0100 0010	42	12-0-9-2
N2	1010 0010	A2	0100 0011	43	12-0-9-3
N3	1010 0011	A3	0100 0100	44	12-0-9-4
N4	1010 0100	A4	0100 0101	45	12-0-9-5
N5	1010 0101	A5	0100 0110	46	12-0-9-6
N6	1010 0110	A6	0100 0111	47	12-0-9-7
N7	1010 0111	A7	0100 1000	48	12-0-9-8
N8	1010 1000	A8	0100 1001	49	12-8-1
N9	1010 1001	A9	0101 0001	51	12-11-9-1
N10	1010 1010	AA	0101 0010	52	12-11-9-2
N11	1010 1011	AB	0101 0011	53	12-11-9-3
N12	1010 1100	AC	0101 0100	54	12-11-9-4
N13	1010 1101	AD	0101 0101	55	12-11-9-5
N14	1010 1110	AE	0101 0110	56	12-11-9-6
N15	1010 1111	AF	0101 0111	57	12-11-9-7
N16	1011 0000	B0	0101 1000	58	12-11-9-8
N17	1011 0001	B1	0101 1001	59	11-8-1
N18	1011 0010	B2	0110 0010	62	11-0-9-2
N19	1011 0011	B3	0110 0011	63	11-0-9-3
N20	1011 0100	B4	0110 0100	64	11-0-9-4
N21	1011 0101	B5	0110 0101	65	11-0-9-5
N22	1011 0110	B6	0110 0110	66	11-0-9-6
N23	1011 0111	B7	0110 0111	67	11-0-9-7
N24	1011 1000	B8	0110 1000	68	11-0-9-8
N25	1011 1001	B9	0110 1001	69	0-8-1
N26	1011 1010	BA	0111 0000	70	12-11-0
N27	1011 1011	BB	0111 0001	71	12-11-0-9-1
N28	1011 1100	BC	0111 0010	72	12-11-0-9-2
N29	1011 1101	BD	0111 0011	73	12-11-0-9-3
N30	1011 1110	BE	0111 0100	74	12-11-0-9-4
N31	1011 1111	BF	0111 0101	75	12-11-0-9-5

ASCII to EBCDIC Translation Chart

ASCII Collating Sequence Order (cont)

SYMBOL	ASCII BINARY		HEX	EBCDIC BINARY		HEX	CARD CODE
N32	1100	0000	C0	0111	0110	76	12-11-0-9-6
N33	1100	0001	C1	0111	0111	77	12-11-0-9-7
N34	1100	0010	C2	0111	1000	78	12-11-0-9-8
N35	1100	0011	C3	1000	0000	80	12-0-8-1
N36	1100	0100	C4	1000	1010	8A	12-0-8-2
N37	1100	0101	C5	1000	1011	8B	12-0-8-3
N38	1100	0110	C6	1000	1100	8C	12-0-9-4
N39	1100	0111	C7	1000	1101	8D	12-0-8-5
N40	1100	1000	C8	1000	1110	8E	12-0-8-6
N41	1100	1001	C8	1000	1111	8F	12-0-8-7
N42	1100	1010	CA	1001	0000	90	12-11-8-1
N43	1100	1011	CB	1001	1010	9A	12-11-8-2
N44	1100	1100	CC	1001	1011	9B	12-11-8-3
N45	1100	1101	CD	1001	1100	9C	12-11-8-4
N46	1100	1110	CE	1001	1101	9D	12-11-8-5
N47	1100	1111	CF	1001	1110	9F	12-11-8-6
N48	1101	0000	D0	1001	1111	0F	12-11-8-7
N49	1101	0001	D1	1010	0000	A0	11-0-8-1
N50	1101	0010	D2	1010	1010	AA	11-0-8-2
N51	1101	0011	D3	1010	1011	AB	11-0-8-3
N52	1101	0100	D4	1010	1100	AC	11-0-8-4
N53	1101	0101	D5	1010	1101	AD	11-0-8-5
N54	1101	0110	D6	1010	1110	AE	11-0-8-6
N55	1101	0111	D7	1010	1111	AF	11-0-8-7
N56	1101	1000	D8	1011	0000	B0	12-11-0-8-1
N57	1101	1001	D9	1011	0001	B1	12-11-0-1
N58	1101	1010	DA	1011	0010	B2	12-11-0-2
N59	1101	1011	DB	1011	0011	B3	12-11-0-3
N60	1101	1100	DC	1011	0100	B4	12-11-0-4
N61	1101	1101	DD	1011	0101	B5	12-11-0-5
N62	1101	1110	DE	1011	0110	B6	12-11-0-6
N63	1101	1111	DF	1011	0111	B7	12-11-0-7
G0	1110	0000	E0	1011	1000	B8	12-11-0-8
G1	1110	0001	E1	1011	1001	B9	12-11-0-9
G2	1110	0010	E2	1011	1010	BA	12-11-0-8-2
G3	1110	0011	E3	1011	1011	BB	12-11-0-8-3
G4	1110	0100	E4	1011	1100	BC	12-11-0-8-4
G5	1110	0101	E5	1011	1101	BD	12-11-0-8-5
G6	1110	0110	E6	1011	1110	BE	12-11-0-8-6
G7	1110	0111	E7	1011	1111	BF	12-11-0-8-7
G8	1110	1000	E8	1100	1010	CA	12-0-9-8-2
G9	1110	1001	E9	1100	1011	CB	12-0-9-8-3
G10	1110	1010	EA	1100	1100	CC	12-0-9-8-4
G11	1110	1011	EB	1100	1101	CD	12-0-9-8-5
G12	1110	1100	EC	1100	1110	CE	12-0-9-8-6
G13	1110	1101	ED	1100	1111	CF	12-0-9-8-7
G14	1110	1110	EE	1101	1010	DA	12-11-9-8-2
G15	1110	1111	EF	1101	1011	DB	12-11-9-8-3
G16	1111	0000	F0	1101	1100	DC	12-11-9-8-4
G17	1111	0001	F1	1101	1101	DD	12-11-9-8-5
G18	1111	0010	F2	1101	1110	DE	12-11-9-8-6
G19	1111	0011	F3	1101	1111	DF	12-11-9-8-7
G20	1111	0100	F4	1110	1010	EA	11-0-9-8-2
G21	1111	0101	F5	1110	1011	EB	11-0-9-8-3
G22	1111	0110	F6	1110	1100	EC	11-0-9-8-4
G23	1111	0111	F7	1110	1101	ED	11-0-9-8-5
G24	1111	1000	F8	1110	1110	EE	11-0-9-8-6
G25	1111	1001	F9	1110	1111	EF	11-0-9-8-7
G26	1111	1010	FA	1111	1010	FA	12-11-0-9-8-2
G27	1111	1011	FB	1111	1011	FB	12-11-0-9-8-3
G28	1111	1100	FC	1111	1100	FC	12-11-0-9-8-4
G29	1111	1101	FD	1111	1101	FD	12-11-0-9-8-5
G30	1111	1110	FF	1111	1110	FE	12-11-0-9-8-6
F0	1111	1111	FF	1111	1111	FF	12-11-0-9-8-7

INDEX

Burroughs

PUBLICATION CHANGE NOTICE

PCN No.: 5001464-001 Date: June 1978

Publication Title: B 7000/B 6000 Series COBOL Reference Manual

Other Affected Publications: None

Supersedes:

Description:

Pages Changed

2-13
5-5
6-87
6-91
6-103
7-79
7-113
7-125
7-127
7-129
13-21
13-23
D-23

Burroughs believes that the information described in this manual is accurate and reliable, and much care has been taken in its preparation. However, no responsibility, financial or otherwise, is accepted 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.

b. LINE-COUNTER. The word LINE-COUNTER is the fixed data-name for a COMPUTATIONAL LINE-COUNTER that is generated for each Report Description in the Report Section to determine the vertical positioning of a report. One LINE-COUNTER is automatically supplied for each report described in the REPORT SECTION if a PAGE LIMIT clause is included in the report description entry.

c. PAGE-COUNTER. The word PAGE-COUNTER is a fixed data-name for a COMPUTATIONAL PAGE-COUNTER that is generated for each Report Description entry in the Report Section for use as a source data item for page numbers within a report group. One PAGE-COUNTER is supplied for each report for which the word PAGE-COUNTER is included as a source data item in a report group description entry.

d. CHECKPOINT-STATUS. The word CHECKPOINT-STATUS is the fixed data-name used by the CHECKPOINT/RERUN facility. See the discussion of CHECKPOINT in Section 7 for a discussion of CHECKPOINT-STATUS.

e. LINAGE-COUNTER. The word LINAGE-COUNTER is a fixed data-name for a COMPUTATIONAL line counter generated by the presence of a LINAGE clause in a File Description. The implicit class of a LINAGE-COUNTER is numeric. The value represented in the LINAGE-COUNTER at any given time is the number of lines advanced within a printed page. One LINAGE-COUNTER is supplied for each file in the FILE SECTION whose FD entry contains a LINAGE clause.

f. TODAYS-DATE. TODAYS-DATE is synonymous with TIME(15) and will return the current date in DISPLAY form in the format "MMDDYY".

g. TIME(n). Various times can be made available to a COBOL program by the use of TIME(n), where n must be an integer ranging from 0 thru 15:

- TIME(0). Returns the current Julian date, in the form "YYDDD", where YY is the last two digits of the year and DDD is the day of the year, in DISPLAY-1 form.
- TIME(1). Returns in COMPUTATIONAL form as an integer value the time of day in sixtieths of a second.
- TIME(2). Returns in COMPUTATIONAL form as an integer value the elapsed processor time of the program in sixtieths of a second.
- TIME(3). Returns in COMPUTATIONAL form as an integer value the elapsed I/O time of the program in sixtieths of a second.
- TIME(4). Returns in COMPUTATIONAL form as an integer value the contents of a 6-bit machine clock which increments every sixtieth of a second.

- TIME(5). Returns the current date in the format "MMDDYY", where MM is the month, DD is the day, and YY is the last two digits of the year, in DISPLAY-1 form.
- TIME(10). This is the same as TIME(0) except the value is returned in DISPLAY form rather than DISPLAY-1.
- TIME(11). This is the same as TIME(1) except the time is in increments of 2.4 microseconds rather than sixtieths of a second.
- TIME(12). This is the same as TIME(2) except the time is in increments of 2.4 microseconds rather than sixtieths of a second.
- TIME(13). This is the same as TIME(3) except the time is in increments of 2.4 microseconds.
- TIME(14). Returns in COMPUTATIONAL form as an integer value, the contents of a 36-bit machine clock which increments every 2.4 microseconds. The contents of the machine clock do not necessarily contain the current time of day and should be used for interval timing purposes only.
- TIME(15). Returns the current date in the format "MMDDYY", where MM is the month, DD is the day, and YY is the last two digits of the year, in DISPLAY form.
- COMPILETIME(n). COMPILETIME(n) is similar to TIME(n). The parameter range n is the same, but COMPILETIME(n) is not dynamic and it returns the values of TIME(n) as they existed at compile time, thus enabling the object program to find out when it was compiled and how long it took.

NOTE: Most special registers have an implicit class of numeric. Numeric special registers and attributes can be used nearly anywhere in the syntax of the **PROCEDURE DIVISION** that a numeric value is acceptable, such as source operands in MOVE, ADD, and SUBTRACT statements.

Mnemonic-Name. The use of mnemonic-names provides a means of relating certain hardware equipment names to problem-oriented names the programmer may wish to use. See the discussion of SPECIAL-NAMES in Section 5.

Index-Name. An index-name is a word with at least one alphabetic character that names an index associated with a specific table (refer to indexing in Section 6). An index is a register, the contents of which represents the character position of the first character of an element of a table with respect to the beginning of the table.

Object-Computer

The function of this paragraph is to describe the computer on which the program is to be executed and to specify core and disk size limitations when using the SORT.

The format for this paragraph consists of two options which are as follows:

Option 1:

```
OBJECT-COMPUTER.  COPY  library-name
                        [ FROM seq.no. ] [ { THRU    } seq.no. ]
                                           { THROUGH }
                        [ REPLACING word-1 BY text-1
                                  [, word-2 BY text-2] ... ] .
```

Option 2:

```
OBJECT-COMPUTER. { B-7700 }
                 { B-6700 }
                 { word-3 }

[, MEMORY SIZE integer { CHARACTERS } ]
                       { WORDS      }
                       { MODULES    }

[, DISK SIZE integer { WORDS   } ]
                     { MODULES }

[, SEGMENT-LIMIT IS integer ]

[,[integer] hardware-name]... .

[, PROGRAM COLLATING SEQUENCE IS alphabet-name ] .
```

For a discussion of the COPY function, refer to Section 8, THE COBOL LIBRARY.

Word-3 is any single COBOL word.

The MEMORY SIZE option is used only in conjunction with a SORT statement. The SORT statement may also specify MEMORY SIZE and will take precedence over the OBJECT-COMPUTER paragraph. When MEMORY SIZE is not specified in the SORT statement and not specified in the OBJECT-COMPUTER paragraph, a default MEMORY SIZE of 12,000 words will be assumed. If this option is used and a SORT statement does not appear in the program, the option will be ignored by the compiler. One module of memory is equivalent to 16,384 words of memory.

The DISK SIZE option is used only in conjunction with the SORT statement. If this clause is omitted in a sort program, DISK SIZE will be assumed to be

900,000 words. If this option is used and a SORT statement does not appear in the program, the option will be ignored by the compiler. One module of disk is equivalent to 1.8 million words of disk.

The SEGMENT-LIMIT clause can be used to control the compiler in segmenting a program. Segmentation normally occurs at the first paragraph name encountered beyond the point at which 1500 words of code have been produced and at the beginning of each SECTION of the PROCEDURE DIVISION. When SEGMENT-LIMIT is specified, the SEGMENT-LIMIT value is used instead of 1500 words. SEGMENT-LIMIT, when specified, is assumed to be in words. The maximum SEGMENT-LIMIT specification is 4095 words.

Segmentation for the initialization code of the WORKING-STORAGE SECTION will occur automatically is the SEGMENT-LIMIT has been reached.

The hardware-name list is for documentation purposes only.

If the ANSI 74 PROGRAM COLLATING SEQUENCE clause is specified, the program collating sequence is the collating sequence associated with the alphabet-name specified in that clause. This collating sequence is used to determine the truth value of non-numeric DISPLAY items comparisons.

If the collating sequence clause is not specified, the native or EBCDIC collating sequence is used to determine the truth value of non-numeric DISPLAY items comparisons.

The ANSI74 system dollar option must be set if the PROGRAM COLLATING SEQUENCE clause is to be used. Refer to Appendix B for a description of the ANSI 74 implementations.

USAGE

The function of the USAGE clause is to specify the format in computer storage of a data item.

```
[USAGE IS] { COMP                 { ASCII
           { COMPUTATIONAL        { DISPLAY
           { COMP-1               { DISPLAY-1
           { COMPUTATIONAL-1      { INDEX
           { COMP-2               { EVENT
           { COMPUTATIONAL-2      { LOCK
           { COMP-4               { CONTROL-POINT
           { COMPUTATIONAL-4      { CP
           { COMP-5
           { COMPUTATIONAL-5
           { INDEX FILE CONTAINS file-name-1 [,file-name-2]...
```

In the absence of any USAGE indication, either explicitly shown in the USAGE clause or within the SIZE clause, USAGE IS DISPLAY is assumed.

The USAGE clause for a report group item can only specify the DISPLAY or DISPLAY-1 option.

The USAGE clause can be written at any level, except CONTROL-POINT, EVENT, INDEX and LOCK may only appear at level 77 or level 01 in WORKING-STORAGE or LOCAL-STORAGE. If the USAGE clause is written at a group level, it applies to each elementary item in the group. Multiple record descriptions for the same file may be declared with contradictory usages. If a group item is described as COMPUTATIONAL, the elementary items are COMPUTATIONAL. The group item itself is not COMPUTATIONAL because it cannot be used in computations.

COMP, COMP-2, COMP-4 and COMP-5 items should not be declared for DISPLAY files (CARD-READER, PRINTER, PUNCH, or REMOTE). The use of such items will cause question marks to print for characters which have no graphic and may cause undesired control characters to be developed.

Care should be exercised in redefining COMP, COMP-1, COMP-4, COMP-5 items on a character basis since the decimal number does not correspond in mapping to its binary equivalent.

The following rules apply when mixing usages:

1. COMP-2 items cannot be subordinate to ASCII, COMP, or DISPLAY-1 group items.
2. COMP, COMP-4 and COMP-5 items can be subordinate to COMP, COMP-1, DISPLAY or DISPLAY-1 group items.

3. COMP-2 items can be subordinate to DISPLAY group items. The compiler, however, may have to insert a 4-bit filler if DISPLAY elementary items follow COMP-2 elementary items.
4. Extreme care must be exercised when moving DISPLAY or DISPLAY-1 group items if either the sending or receiving field contains subordinate COMP, COMP-2, COMP-4 or COMP-5 data.
5. No mixing of USAGE or contradiction of USAGE is permitted subordinate to a group declared CONTROL-POINT, EVENT, INDEX or LOCK. These items may not be subordinate to an item of any other USAGE.
6. Automatic translation of data takes place when moving data between ASCII, COMP-2, DISPLAY, and DISPLAY-1 items. Any character in one character set which is undefined in the other character set is translated into the question mark.
7. INDEX, EVENT, LOCK and CP items must not be declared subordinate to an item of USAGE COMP, COMP-1, COMP-2, COMP-4, COMP-5, ASCII, DISPLAY or DISPLAY-1. INDEX, EVENT, LOCK and CP items may not be mixed in an 01 level. When an array is to contain INDEX, EVENT, LOCK or CP items, USAGE must be declared at the 01 level.
8. Floating-point format numeric literals may only be moved to floating-point COMP-4 or COMP-5 receiving fields.

USAGE is the dominant declaration for internal format representation within the computer system and is defined for use as follows:

a. DISPLAY. The data item is assumed to contain eight-bit-coded EBCDIC characters, six characters for each B 7000/B 6000 word.
b. DISPLAY-1. The data is assumed to contain six-bit-coded BCL characters, eight characters per computer word.
c. ASCII. The data is assumed to contain eight-bit-coded ASCII characters, six characters for each B 7000/B 6000 word. If an array is to contain ASCII characters, the 01 level must be declared ASCII and no subordinate item may declare a usage other than ASCII. No data item within an ASCII array may be declared as numeric or edited numeric.
d. COMPUTATIONAL (COMP). The data item is to be used primarily for arithmetic operations; therefore, it is maintained in a binary coded representation.

    1. COMPUTATIONAL implies a signed item. No further sign specification is required. (An unsigned value may be obtained through the ABS function.)

g. COMPUTATIONAL-4 (COMP-4). This data item will be a single precision internal floating point operand which will occupy one word of memory; however, the item is not necessarily word-aligned.

A PICTURE entry is not permitted for a COMP-4 item.

All comments for COMP items apply equally to COMP-4 items except the size requirements, allowable picture characters, and that COMP-4 is only permissible in an elementary item description.

h. COMPUTATIONAL-5 (COMP-5). This data item will be a double precision internal floating point operand which will occupy two words of memory; however, the item is not necessarily word-aligned.

A PICTURE entry is not permitted for a COMP-5 item.

All comments for COMP items apply equally to COMP-5 items except the size requirements, allowable picture characters, and that COMP-5 is only permissible in an elementary item description.

i. INDEX. An elementary item described with the USAGE IS INDEX clause is called an index data item. If a group item is described with the USAGE IS INDEX clause, the elementary items in the group are all index data items. The group itself is not an index data item and cannot be used in SEARCH or SET statements or in a relational condition. An index data item can be referred to directly only in a PERFORM, SEARCH, or SET statement, or in a relational condition. An index data item can be part of a group referred to in a MOVE or input-output statement, in which case no conversion will take place. The SIZE,SYNCHRONIZED, BLANK WHEN ZERO, JUSTIFIED, PICTURE, and VALUE clauses cannot be used to describe group or elementary items described with the USAGE IS INDEX clause.

Index data items can also be declared subordinate to a usage DISPLAY group item. This offers compatibility with both B 4700 and B 1700 systems COBOL languages and with the ANSI 1968 and ANSI 1974 COBOL standards. The internal representation of all index data names is that of a signed seven-digit four bit character field. This internal representation is the same as the B 4700 and B 1700 COBOL compilers give for index data items.

j. INDEX FILE. DIRECT SWITCH FILES may be declared at the 77 level in WORKING or LOCAL-STORAGE SECTIONS by specifying a usage of INDEX FILE.

Example:

```
77 switch-file-identifier INDEX FILE [CONTAINS file-name-1
     [,file-name-2]  ...]
```

The CONTAINS clause is used to describe which DIRECT files compose the switch. Each file named in the switch must be a DIRECT file and an FD must be provided.
The CONTAINS clause must be present if the DIRECT SWITCH FILE is not received as a parameter. When the DIRECT SWITCH FILE is received as a parameter, or declared in LOCAL-STORAGE, then the RECEIVED clause must be used to indicate that it is RECEIVED BY REFERENCE (name) and the CONTAINS clause must not be specified.

A DIRECT SWITCH FILE identifier can be used any place in the syntax that a DIRECT FILE identifier can be used, namely in OPEN, CLOSE, READ, and WRITE statements and in attribute expressions.

Example:

```
OPEN INPUT SWFL(X)
:
     READ SWFL(X) KEY IS RCINR(X) INTO RCDAREA(X)
```

All reads and writes with DIRECT SWITCH FILES must use a KEY clause if non-serial action is desired, even if all the DIRECT FILES in the switch are declared to be random.

If DIRECT SWITCH FILES are passed as parameters, then the corresponding formal parameter description must be a DIRECT SWITCH FILE. A program which receives a DIRECT SWITCH FILE as a parameter must not have an FD for the files contained in the switch, since these files were described in the program which passed them as parameters.

k. EVENT. Items described with the USAGE IS EVENT clause are used to give the programmer a means of testing and controlling DIRECT input-output operations (i.e., I-O COMPLETE). For asynchronous processing, EVENT's may be used as a common interlock between two or more processes, thus providing an efficient means of correlating the activities of one process with its related processes.

EVENT usage is allowed only on a 77 or 01 level item and, if used at an 01 level, may have a subordinate OCCURS clause. (See the OCCURS clause.) Except for documentary uses of the SIZE clause, no other entries are permitted with an EVENT name.

DATA-BASE SECTION

The purpose of the data-management or data-base system is to create and maintain data. This data is accessed by a COBOL program using an extended set of verbs. Use of these verbs requires very little programming effort, but gives COBOL programs the ability to INQUIRY, CREATE, and MODIFY data stored in a common data-base. The advantages and use of a data-base with the DMSII system are described in the B 7000/B 6000 HOST LANGUAGE REFERENCE MANUAL, Form No. 5001498.

LOCAL-STORAGE SECTION

The optional LOCAL-STORAGE SECTION describes parameters received by a procedure when it is invoked.

LD local-storage-name.

The LD entry is followed by item descriptions as used in the WORKING-STORAGE SECTION.

Local storage is associated with a specific procedure by the USE statement mentioning the local-storage-name. The local-storage-name must be unique. An LD entry is required for each procedure that receives data as parameters (i.e., the USING clause is used in both the invocation of the procedure and the USE statement in the section header).

SOURCE ITEM \ DESTINATION ITEM			GROUP					ELEMENTARY																				
								NUMERIC										NUMERIC-EDITED		ALPHA-NUMERIC			ALPHA-NUMERIC EDITED			ALPHA-BETIC		
										INTEGER				NON-INTEGER														
			COMP	COMP-2	DISPLAY-1	ASCII	DISPLAY	COMP-4/COMP-5	DMS FIELD	COMP	COMP-2	DISPLAY-1	DISPLAY	COMP	COMP-2	DISPLAY-1	DISPLAY	DISPLAY-1	DISPLAY	DISPLAY-1	ASCII	DISPLAY	DISPLAY-1	ASCII	DISPLAY	DISPLAY-1	ASCII	DISPLAY
GROUP		COMP	16	16	17	17	17	*	*	*	16	17	17	*	16	17	17	17	17	17	17	17	17	17	17	17	17	17
		COMP-2	16	2	2	2	2	*	*	*	2	2	2	*	2	2	2	2	2	2	2	2	2	2	2	2	2	2
		DISPLAY-1	16	2	2	2	2	*	*	*	2	2	2	*	2	2	2	2	2	2	2	2	2	2	2	2	2	2
		ASCII	16	2	2	2	2	*	*	*	2	2	2	*	2	2	2	2	2	2	2	2	2	2	2	2	2	2
		DISPLAY	16	2	2	2	2	*	*	*	2	2	2	*	2	2	2	2	2	2	2	2	2	2	2	2	2	2
LITERAL		FLOATING-POINT	*	*	*	*	*	5	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
		"UNDIGIT"	*	6	*	*	*	*	*	*	6	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
		NUMERIC INTEGER	*	25	25	25	25	3	7	3	1	1	1	3	1	1	1	20	20	25	25	25	26	26	26	*	*	*
		NUMERIC NON-INTEGER	*	25	25	25	25	3	7	3	1	1	1	3	1	1	1	20	20	*	*	*	*	*	*	*	*	*
		ALPHANUMERIC	*	23	23	23	23	*	*	*	9	9	9	*	9	9	9	21	21	23	23	23	24	24	24	23	23	23
ELEMENTARY		COMP-4/COMP-5	*	14	14	*	14	19	7	8	12	12	12	8	12	12	12	22	22	14	*	14	15	*	15	*	*	*
		DMS FIELD	*	*	*	*	*	4	7	4	4	4	4	4	4	4	4	22	22	*	*	*	*	*	*	*	*	*
	NUMERIC	INTEGER COMP	*	14	14	*	14	19	7	8	12	12	12	8	12	12	12	22	22	14	*	14	15	*	15	*	*	*
		INTEGER COMP-2	16	2	2	*	2	3	7	3	1	1	1	3	1	1	1	20	20	14	*	14	15	*	15	*	*	*
		INTEGER DISPLAY-1	16	2	2	*	2	3	7	3	1	1	1	3	1	1	1	20	20	14	*	14	15	*	15	*	*	*
		INTEGER DISPLAY	16	2	2	*	2	3	7	3	1	1	1	3	1	1	1	20	20	14	*	14	15	*	15	*	*	*
		NON-INTEGER COMP	*	*	*	*	*	18	7	8	12	12	12	8	12	12	12	22	22	*	*	*	*	*	*	*	*	*
		NON-INTEGER COMP-2	16	2	2	*	2	3	7	3	1	1	1	3	1	1	1	20	20	*	*	*	*	*	*	*	*	*
		NON-INTEGER DISPLAY-1	16	2	2	*	2	3	7	3	1	1	1	3	1	1	1	20	20	*	*	*	*	*	*	*	*	*
		NON-INTEGER DISPLAY	16	2	2	*	2	3	7	3	1	1	1	3	1	1	1	20	20	*	*	*	*	*	*	*	*	*
	NUMERIC-EDITED	DISPLAY-1	16	2	2	*	2	*	*	*	*	*	*	*	*	*	*	*	*	2	2	2	13	13	13	*	*	*
		DISPLAY	16	2	2	*	2	*	*	*	*	*	*	*	*	*	*	*	*	2	2	2	13	13	13	*	*	*
	ALPHA-NUMERIC	DISPLAY-1	16	2	2	2	2	10	11	10	9	9	9	10	9	9	9	21	21	2	2	2	13	13	13	2	2	2
		ASCII	16	2	2	2	2	10	11	10	9	9	9	10	9	9	9	21	21	2	2	2	13	13	13	2	2	2
		DISPLAY	16	2	2	2	2	10	11	10	9	9	9	10	9	9	9	21	21	2	2	2	13	13	13	2	2	2
	ALPHA-NUMERIC EDITED	DISPLAY-1	16	2	2	2	2	*	*	*	*	*	*	*	*	*	*	*	*	2	2	2	13	13	13	2	2	2
		ASCII	16	2	2	2	2	*	*	*	*	*	*	*	*	*	*	*	*	2	2	2	13	13	13	2	2	2
		DISPLAY	16	2	2	2	2	*	*	*	*	*	*	*	*	*	*	*	*	2	2	2	13	13	13	2	2	2
	ALPHA-BETIC	DISPLAY-1	16	2	2	2	2	*	*	*	*	*	*	*	*	*	*	*	*	2	2	2	13	13	13	2	2	2
		ASCII	16	2	2	2	2	*	*	*	*	*	*	*	*	*	*	*	*	2	2	2	13	13	13	2	2	2
		DISPLAY	16	2	2	2	2	*	*	*	*	*	*	*	*	*	*	*	*	2	2	2	13	13	13	2	2	2

Figure 7-10. Valid MOVE Statement Combinations

The following rules describe the valid combinations of sending and receiving fields shown in figure 7-10. These rules apply in addition to the standard alignment rules for destinations as explained in the discusssion of the JUSTIFIED clause.

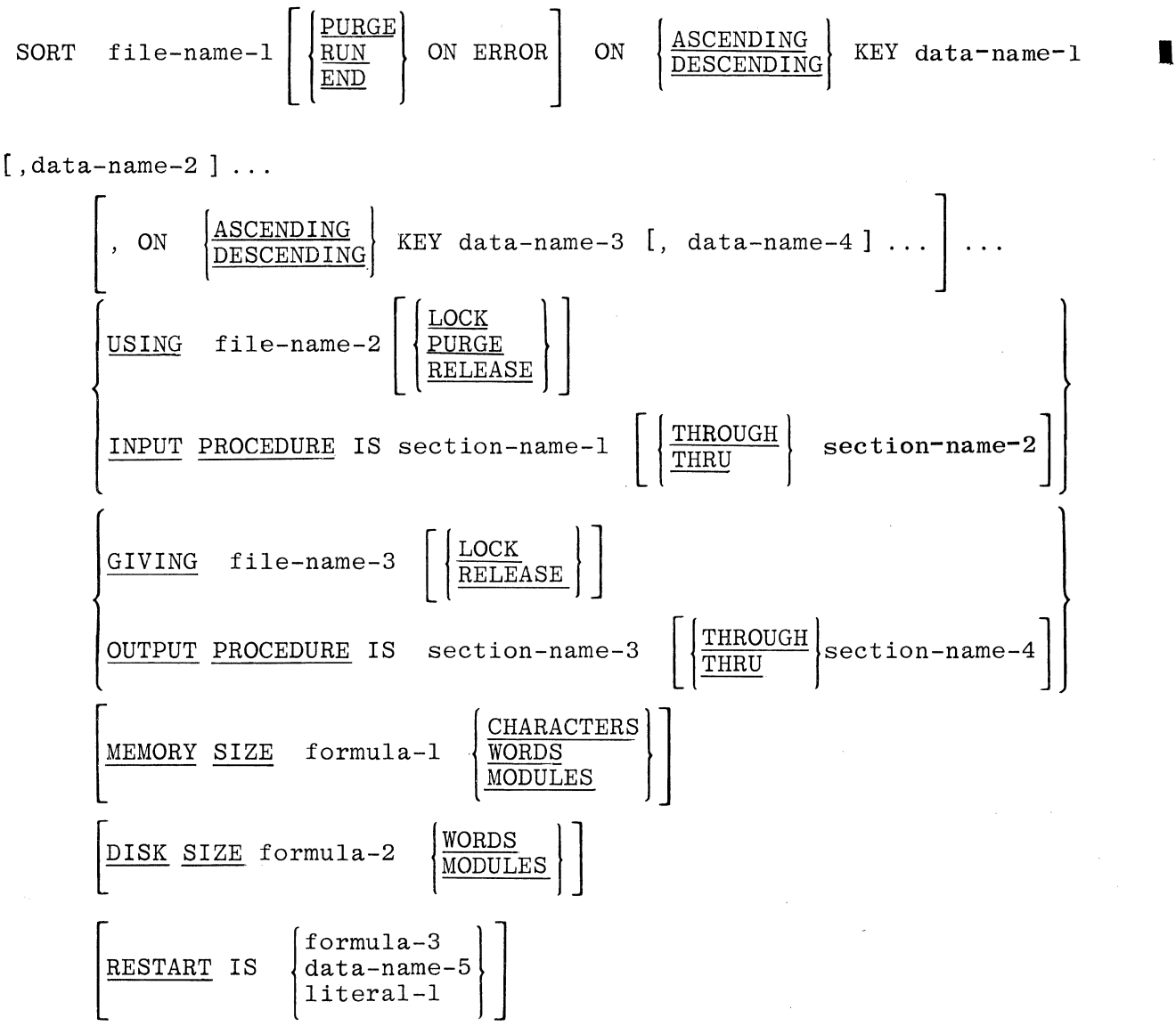
* Illegal Move

1. Numeric-decimal move; absolute value moved if destination is described as unsigned. Completion of operation guarantees that zones and sign of destination, if any, are valid. If digits are greater than 9, they may not be moved unchanged from the source.
2. Alphanumeric move; left justified with truncation or space fill (zero fill if the destination is COMP-2) on the right (except if the destination is described with the JUSTIFIED clause). If the usage of the source and destination are not the same, translation occurs using the standard MCP translate tables. The source is considered as having a category of alphanumeric. If the source is a group item, the destination is considered as having a category of alphanumeric (any editing or decimal point is ignored).
3. Numeric move; the decimal source is converted to binary. "UNDIGITS" existing in the source are changed to values less than 10. If the destination is COMP-4 or COMP-5, the source value will be approximated in binary floating-point.
4. Numeric move; the bit pattern of the source is considered to be an unsigned integer operand, and is extended to double precision binary or converted to decimal if necessary.
5. Numeric move; decimal value converted to nearest binary floating-point approximation, and adjusted to single or double precision.
6. "UNDIGIT" move; right justified, zero fill on left.
7. Numeric move; the source is converted to a binary integer if necessary. The destination is considered to be an unsigned binary integer operand. High-order bits of the source may be truncated.
8. Numeric move; the binary source value is adjusted to the precision and scale of the destination. If the source value is floating-point, the source value is integerized.
9. Numeric move; the source is considered as a numeric unsigned integer, and is moved as described in rule-1.
10. Numeric move; the source is considered as a numeric unsigned integer, and is moved as described in rule-3.

SORT

The SORT statement is used to create a sort-file by executing input procedures or by transferring records from another file, to sort the records in the sort-file on a set of specified keys and, in the final phase of the operation, to make available each record from the sort-file, in sorted order, to an output procedure or to an output file.

The format for the SORT statement is as follows:



More than one SORT statement may appear in a program. SORT statements can appear anywhere in the PROCEDURE DIVISION, except within INPUT and OUTPUT procedures associated with a SORT statement or within the DECLARATIVES.

File-name-1 must have a sort-file description in the DATA DIVISION. File-name-2 and file-name-3 must be described in a file description entry, not in a sort-file description entry.

The keys are listed from left to right in the SORT statement in order of significance, without regard to how they occur within the record. Data-name-1 is the major key followed in descending significance by additional keys if any.

a. When a SORT file-name-1 PURGE clause is used, PURGE implies that on a parity error, the bad record will be skipped and SORTing will continue.
b. When a SORT file-name-1 RUN clause is used, RUN implies that on a parity error, the bad record will be SORTed.
c. When a SORT file-name-1 END clause is used, END implies that the SORT will be DSed on encountering a parity error.
d. When an ASCENDING clause is used, the sorted sequence will be from the lowest value of key to highest value.
e. When a DESCENDING clause is used, the sorted sequence will be from the highest value of key to lowest value.
f. Both ASCENDING and DESCENDING ekys can be used in one SORT statement.
g. When a USING file-name-2 LOCK clause is used, LOCK implies that the file will be LOCKed at the end of the SORT.
h. When a USING file-name-2 PURGE clause is used, PURGE implies close with PURGE.
i. When a USING file-name-2 RELEASE clause is used, RELEASE implies close with RELEASE.
j. When a GIVING file-name-3 LOCK clause is used, LOCK implies that the file will be closed with LOCK at the end of the SORT.
k. When a GIVING file-name-3 RELEASE clause is used, RELEASE implies close with RELEASE.

NOTE: Items a thru c and g thru k can be implemented only when the B2500 system dollar option is set.

Every data-name appearing in the KEY clause must be described under the DATA DIVISION entry for the sort-file-name, and these KEY items are subject to the following rules:

USE

The USE statement is employed in certain declaratives and has three different functions:

a. It can specify supplemental procedures for I/O error and label-handling.
b. It can specify procedures to be employed in a parallel processing environment, and
c. It can specify interrupt procedures.

The format for the USE statement has five options which are as follows:

Option 1:

```
USE AFTER STANDARD ERROR PROCEDURE ON { INPUT-OUTPUT                          }.
                                      { I-O                                   }
                                      { INPUT                                 }
                                      { file-name-1  [,file-name-2] ...       }
```

Option 2:

```
USE AFTER RECORD SIZE ERROR ON { file-name-1 [,file-name-2] ... }.
```

Option 3:

```
USE { BEFORE } STANDARD { BEGINNING } { REEL }
    { AFTER  }          { ENDING    } { FILE }

LABEL PROCEDURE ON { INPUT-OUTPUT                    }.
                   { I-O                             }
                   { INPUT                           }
                   { OUTPUT                          }
                   { file-name-1 [,file-name-2]      }
```

Option 4:

```
USE { EXTERNAL { identifier-1    } AS PROCEDURE }
    { AS GLOBAL { mnemonic-name }               }

  [ ; WITH { file-name       }    [, { file-name       }] ...
           { local-storage-name }     { local-storage-name }

    ; USING identifier-2 [, identifier-3] ... ] .
```

Option 5:

```
USE AS INTERRUPT PROCEDURE.
```

Option 6:

```
USE AFTER STANDARD { EXCEPTION } PROCEDURE ON { file-name-1 [, file-name-2]... }.
                   { ERROR     }              { INPUT                          }
                                              { OUTPUT                         }
                                              { I-O                            }
                                              { EXTEND                         }
```

USE

A USE statement, when present, must immediately follow a section header in the Declaratives Section. The remainder of the section must consist of zero, one or more procedural paragraphs that define the procedures to be used.

For Example:

```
section-name SECTION.   USE   statement.
(paragraph-name . (sentence) ... ) ...
```

The USE statement itself is never executed; rather, it defines the conditions calling for the execution of the USE procedures. Only CALL, EXECUTE, PROCESS or RUN statements may reference section-name. There must be no reference to the "main body" of the PROCEDURE DIVISION from within a USE section.

Within a given format, a file-name must not be referred to, implicitly or explicitly, in two or more identical USE statements. The same file-name can appear in a different option of the USE statement. However, appearance of a file-name in a USE statement must not cause the simultaneous request for execution of more than one USE declarative. A file-name may not represent a sort-file. Within a USE procedure, there must not be any reference to any non-declarative procedures. Conversely, in the non-declarative portion there must be no reference (other than by a CALL, EXECUTE, PROCESS or RUN) to procedure-names that appear in the declarative portion. A GO TO or PERFORM statement in an Option 1, Option 2, or Option 3 declarative may reference a paragraph-name in any other Option 1, Option 2, or Option 3 declarative section.

The designated procedures are executed by the input-output system at the appropriate time as follows:

a. In Option 1, after completing the standard input-output error routine on files assigned to DISK, TAPE, DISKPACK, or REMOTE.

b. In Option 2, after making available any record from TAPE of a BLOCK which is less than the specified block size.

c. In Option 3, before or after a beginning or ending input label-check procedure is executed.

d. Before a beginning or ending output label is created.

e. After a beginning or ending output label is created, but before it is written.

f. Before or after a beginning or ending input-output label-check procedure is executed.

When Option 2 is used:

a. The RECORD SIZE ERROR is applicable to input TAPE files only.
b. The RECORD SIZE ERROR may not be specified if the file is declared to contain variable length records or blocks.
c. The ON phrase must specify file-name-1, file-name-2...instead of INPUT.

When Option 3 is used:

a. If the file-name option is used, the file description entry for file-name must not specify LABEL RECORDS ARE OMITTED.
b. If the INPUT, OUTPUT, or I-O clause is used, the procedures will not be executed for any INPUT, OUTPUT, or I-O file whose file description entry specifies LABEL RECORDS ARE OMITTED.
c. If BEGINNING or ENDING is not included, the designated procedure will be executed for both beginning and ending labels.
d. If REEL or FILE is not included, the designated procedure will be executed for the appropriate REEL and FILE labels.
e. Option 3 is applicable only to files assigned to tape.

Within the procedures in a USE declarative in which the USE sentence contains either the INPUT or the OUTPUT option, references to label items must be qualified.

Option 4 enables untyped procedures or subroutines to be declared GLOBAL in the same way as they are declared EXTERNAL. This statement is placed in the header of a section to be used as a task.

If the EXTERNAL phrase is used, it identifies the separately compiled program which is to be used as the task when this section is referenced. In addition, there must be no paragraphs in this section when this phrase is employed. Identifier-1 must be defined in the WORKING-STORAGE SECTION such that its value may be a program-name. If the mnemonic-name is used, it must be defined in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION.

Local-storage-name must be defined in the LOCAL-STORAGE SECTION and must be unique among USE statements. A local-storage-name must be present if the USING phrase is present.

If the EXTERNAL phrase is used, the USING phrase is included in the USE statement if, and only if, there is a USING phrase in the PROCEDURE DIVISION header of the referenced, separately compiled program. The number, type and order of the operands in the two USING phrases must then be identical.

When GLOBAL is specified, an untyped procedure must exist in the host with the same name as the section to which the USE is attached.

Identifier-2, identifier-3, ..., must be uniquely defined as files in the FILE SECTION or as level 01 or 77 items in the LOCAL-STORAGE associated with the USE procedure. They may describe any combination of data items, control-point items, event items, index data items, or lock items.

Option 5 of the USE statement is used to specify a declarative as an interrupt procedure.

An interrupt procedure provides a means of interrupting a process when an event-item attached to that procedure is caused. The USE statement may thus be used to declare such interrupt procedures.

Statements which are to be executed when the event is caused and the interrupt procedure is allowed, must follow the USE statement.

Option 6 of the USE statement is used as part of the ANSI 74 Sequential I/O Module specification; thus, requiring that the ANSI74 system dollar option be set.

The words EXCEPTION and ERROR are synonymous.

Specifically, this USE routine option will function the same as Option 1 with the following exception: READ statements may have optional AT END clauses's. This means if a READ statement which contains no AT END clause gets an end-of-file exception, the appropriate USE routine will be executed. If no USE routine exists, the program will be terminated.

WAIT

The WAIT or AWAIT statement is used as part of the user's control of direct access input-output operations and for communication between processes in an asynchronous processing environment.

The format of the WAIT statement is as follows:

Option 1:

```
WAIT  { formula          }
      { event-identifier }
      { INTERRUPT        }
```

Option 2:

```
WAIT control-point-identifier ([subscript,] EXCEPTIONEVENT)
```

Option 3:

```
WAIT [AND RESET]  [formula]
    event-identifier-1 [, event-identifier-2]...
    [GIVING data-name-1]
```

Option 4:

```
WAIT  area-identifier [ON EXCEPTION  statement  [ELSE  statement]]
```

All parentheses are required. When a subscript is specified, a maximum of one subscript or index is allowed and it must be followed by a comma.

Option 1 of the WAIT statement containing an event-identifier will cause processing to be suspended until event-identifier has been caused explicitly by a CAUSE statement, or implicitly by option 3 of the READ statement or option 4 of the WRITE statement. The EVENT is not automatically reset.

If formula is used in option 1, the program will be suspended for the number of seconds (or fraction thereof) specified. Formula must be greater than or equal to zero. A control-point-attribute may be specified as part of formula. However, the formula must then by enclosed in parenthesis and the specified attribute must be typed INTEGER or typed REAL.

The INTERRUPT phrase causes execution of this task to be suspended until at least one of its interrupt procedures has been executed.

Option 2 is identical to an option 1 with an event-identifier specified. The event-identifier in this case is the EXCEPTIONEVENT task attribute associated with the control-point-identifier.

Option 3 causes a WAIT for the number of seconds specified by formula (if specified) or until one of the event-identifiers has been CAUSE'd. If the AND RESET phrase was specified, the event-identifier which caused the WAIT to be terminated will be RESET. If the number of seconds specified by "formula" elapse before one of the event-identifiers is CAUSE'd, the AND RESET has no effect. The GIVING phrase provides an integer value to indicate by which method the WAIT was terminated. Thus, the data-name will contain a 1 if the wait was terminated because the number of seconds specified in "formula" had elapsed, a 2 if event-identifier-1 was CAUSE'd, a 3 if event-identifier-2 was CAUSE'd, etc. If a "formula" was not specified, then data-name will contain a 1 if the WAIT was terminated because event-identifier-1 was CAUSE'd, a 2 if event-identifier-2 was CAUSE'd, etc.

Option 4 of the WAIT statement will cause processing to be suspended until the DIRECT input-output operation (option 3 of the READ statement or option 5 of the WRITE) is complete. Area-name is the name of an array specified as a direct array by use of the RECORD AREA clause.

The ON EXCEPTION phrase of option 4 offers a means of detecting abnormal file conditions when DIRECT I/O operations are used. The specific condition may be determined by using the attributes described in the B 7000/B 6000 INPUT/OUTPUT SUBSYSTEM REFERENCE MANUAL, Form No. 5001779.

The following are examples of valid WAIT statements:

```
WAIT CP-I(5,EXCEPTIONEVENT).
WAIT MYSELF(EXCEPTIONEVENT).
WAIT EVENT-NAME.
WAIT X DIV 2.
WAIT (X).
WAIT CP-I(3,TASKVALUE).
WAIT (MYSELF(TASKVALUE)).
WAIT INTERRUPT.
WAIT 35 EVENT-1, EVENT-2, EVENT-3, GIVING XYZ.
WAIT AND RESET 10 EVENT-6.
WAIT BUF-NAME ON EXCEPTION STOP "ERROR".
```

SEQ

This option activates the resequencing of source-language output files (i.e., NEWTAPE, pass-1 listing, and pass-2 listing). The SEQ option is normally associated with a beginning sequence number and an increment which is added for subsequent source-language images. These two integer values may be specified on a $ card prior to the $ card which activates the SEQ, on the $ card which activates the SEQ, or the default value (10) supplied automatically by the compiler may be used. As long as SEQ is set, the beginning sequence number is incremented by the increment for each source-language record, and that value is retained while SEQ is not set. Thus, setting SEQ does not restore the beginning sequence number. The beginning sequence number is initialized when an unsigned integer is found in any $ card (with or without SEQ set). The increment is initialized by specifying a plus sign (+) followed by an integer in any $ card (with or without SEQ set). The two integer values need not be specified in the same $ card nor even on the $ card specifying the SEQ option.

SEQERR

With this option, the compiler will print warning messages for sequence errors in the source-language input. At the end of a compilation which had sequence errors, a code file will not be created. Thus, sequence errors which print as warnings behave as though they were fatal when SEQERR is set. The settings of NEWSEQERR, SEQ, SPEC and CHECK have no effect on the setting of SEQERR.

SINGLE

This option causes the compiler output listing to be single spaced. SINGLE is automatically set unless the option DOUBLESPACE is set when the compiler is compiled.

SPEC

The SPEC option suppresses printing of warning messages, sequence error messages, printing of the expansion of the DMS INVOKE statement, and the printing of the list of elementary items involved in a CORRESPONDING option.

STACK

This option causes relative stack addresses and the name of the associated item to be printed on the output listing. The STACK option will become active when STACK is set or both LIST and CODE are set.

STATISTICS

It is possible to obtain statistics which reveal the characteristics of a COBOL object job. Statistics are accumulated for a program when the STATISTICS dollar option is set. This option may not be changed after the compiler has encountered the beginning of the IDENTIFICATION DIVISION. When this option is set, the compiler will include code to determine how many times each paragraph is entered and how much time is spent executing the instructions comprising each paragraph. The STATISTICS dollar option can only be set for a compilation at level 2.

Each paragraph has a unique number. This number is printed on the right-hand side of the compiler listing and corresponds to one line of output on the system summary of the statistics. This summary is written on the job's diagnostic file (the file on which program dumps appear). A simple example follows:

```
PROCEDURE DIVISION.
P1.
                                                P1 IS #0001
   OPEN INPUT CARD-FILE.
P2.
                                                P2 IS #0002
   READ CRD AT END GO TO DONE.
   IF FLD=0 THEN PERFORM ZERO-
   RECORD.
P3.
                                                P3 IS #0003
   MOVE 5 TO R-CLASS.
```

The output of the statistics summary would look like the following example:

STATISTICS (Cont)

BLOCK	FREQ	TOTAL TIME	AVG TIME
MAIN	1	0.132267	0.132267
1	1	600	600
2	1000	500	0.500000
3	366	36600	100
.	.	.	.
.	.	.	.
.	.	.	.

The column labeled BLOCK specifies the paragraph, as numbered on the source listing, for which the line of output applies. The line labeled MAIN is the time necessary to initialize user data areas and construct the stack.

The data listed under the heading FREQ reflects the number of times the paragraph was entered. Paragraphs never entered are not listed.

The data under TOTAL TIME is the total processor time spent processing the paragraph. The column AVG TIME is equal to the value of TOTAL TIME divided by FREQ. One should note that times printed in both of these columns without a decimal point are times in microseconds. Thus the total time spent in P1 would be 600 microseconds; but, for P2 it would be one half of a second.

A statistics summary is produced at END-OF-TASK, or when the program is DS-ed.

TIME	This option causes the compiler to print the normal heading and footing of the compilation listing even though the option LIST was never set.
VOID	This option causes all source-language input (primary and secondary), except $ cards, to be ignored until the option becomes not set.
VOIDT	This option causes all secondary source-language input except $ cards to be ignored until the option becomes not set. This option is not active unless the MERGE option is set.
XREF	When this option is set, a cross-reference listing will be produced of declarations and uses of all data-names, file-names, condition-names, etc. Operation of XREF is in no way dependent on the setting of LIST. By use of SET, POP, and RESET, the XREF operation may be restricted to portions of the program.
$	This option causes all $ control cards to be listed.
Integer	When an integer value (not preceded by the symbol +) appears on a $ control card, the integer value is used as the beginning sequence number for the operation of the SEQ operation. This is not a settable option.
+ Integer	When an integer value with a preceding plus symbol (with or without intervening blank spaces) appears in a $ control card, the integer value will be used as the increment for operation of the SEQ $ option. This is not a settable option.
Non-Numeric Literal	When a non-numeric literal is specified in a $ control card, the literal will be used for the replacement operation performed by the NEWID $ option.

Option 1:

```
STOP RUN
```

Option 2:

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

Option 1:

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

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

    [; ON SIZE ERROR statement [ELSE statement]]
```

Option 2:

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

    FROM { literal-m    } GIVING
         { identifier-m }

       identifier-n [ROUNDED] [ , identifier-o [ROUNDED]] ...

    [; ON SIZE ERROR statement [ELSE statement]]
```

Option 3:

```
SUBTRACT { CORR          } identifier-1
         { CORRESPONDING }

      FROM identifier-2 [ROUNDED]

    [; ON SIZE ERROR statement [ELSE statement]]
```

```
TERMINATE report-name-1 [, report-name-2]...
```

```
UNLOCK { identifier      }...
       { lock-identifier }
       { event-identifier }
```

Option 1:

```
USE AFTER STANDARD ERROR PROCEDURE ON { INPUT-OUTPUT                  }.
                                      { I-O                           }
                                      { INPUT                         }
                                      { file-name-1 [, file-name-2] ... }
```

Option 2:

```
USE AFTER RECORD SIZE ERROR ON { file-name [, file-name-2] ... }.
```

Option 3:

```
USE { BEFORE } STANDARD { BEGINNING } { REEL }
    { AFTER  }          { ENDING    } { FILE }

    LABEL PROCEDURE ON { INPUT-OUTPUT                          }.
                       { I-O                                   }
                       { INPUT                                 }
                       { OUTPUT                                }
                       { file-name-1 [, file-name-2]...        }
```

Option 4:

```
USE { EXTERNAL          { identifier-1   } AS PROCEDURE }
    { AS GLOBAL PROCEDURE { mnemonic-name }             }

  [ ; WITH { local-file-name   } [ , { local-file-name   } ] ...
           { local-storage-name }    { local-storage-name }

    ; USING identifier-2 [, identifier-3]... ].
```

Option 5:

```
USE AS INTERRUPT PROCEDURE.
```

Option 6:

```
USE AFTER STANDARD {EXCEPTION} PROCEDURE ON { file-name-1 [,file-name-2]... }
                   {ERROR    }              { INPUT                         }
                                            { OUTPUT                        }
                                            { I-O                           }
                                            { EXTEND                        }
```

Option 1:

```
WAIT { formula          }
     { event-identifier }
     { INTERRUPT        }
```

Option 2:

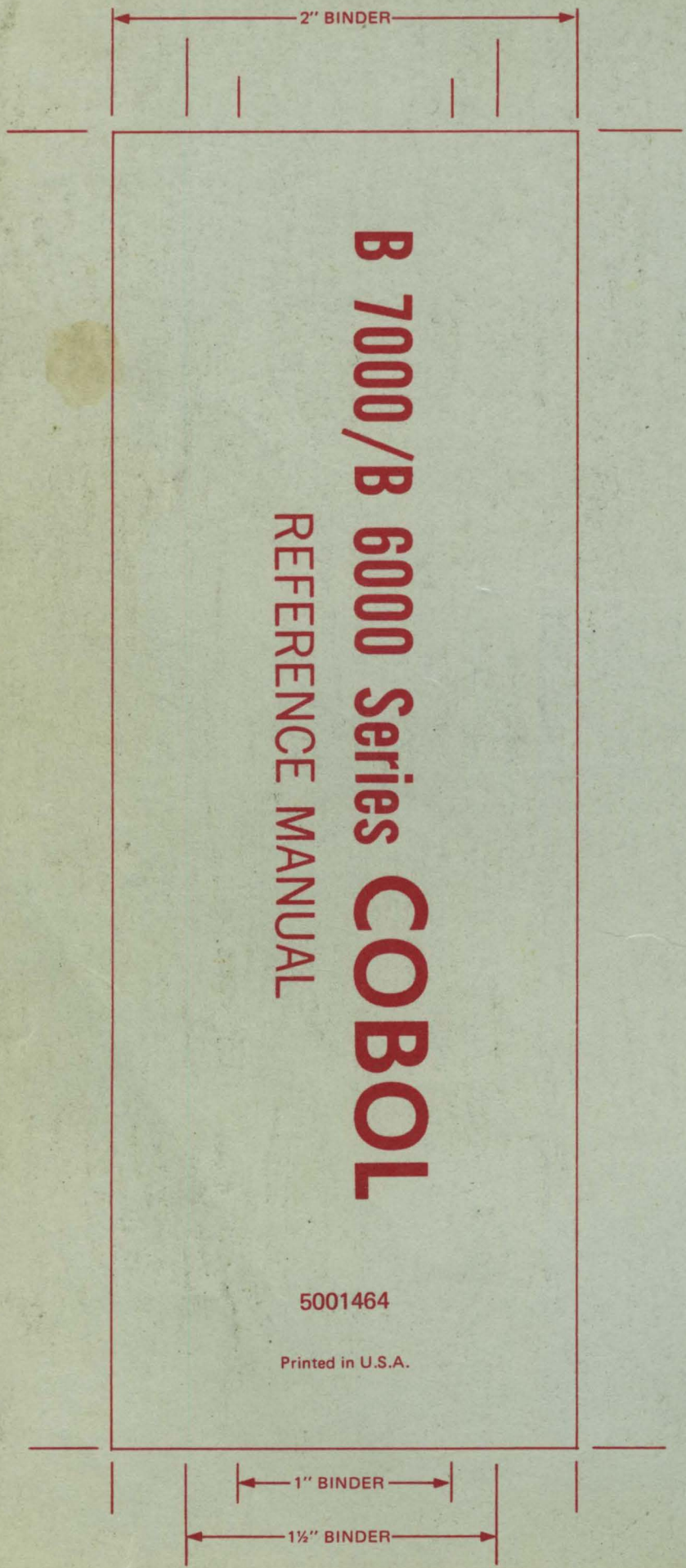
```
WAIT control-point-identifier ([subscript,] EXCEPTIONEVENT)
```

Option 3:

```
WAIT [AND RESET] [formula]
   event-identifier-1 [, event-identifier-2]...
   [GIVING data-name-1]
```

Option 4:

```
WAIT area-identifier [ON EXCEPTION statement [ELSE statement]]
```



Printed in U.S.A. August 1977 5001464