MRX/OS COBOL Level 1

Reference Manual

2202.002

MEMOREX

Computer System Products

November 1972 Edition

PREFACE

This document describes the Memorex implementation of American National Standard (ANS) COBOL, and all Memorex extensions to that standard.

In this document the term MRX COBOL means the Memorex implementation of ANS COBOL and all extensions to it. There are two types of extensions.

1. Those that represent features not specified by ANS COBOL
2. Those that represent an easing of the strict ANS COBOL rules and allow for greater programming convenience

All such extensions are printed on a shaded background for the convenience of users who wish strict conformance with the standard. Use of features that are extensions to the standard may result in incompatibilities between the implementation represented by this document and other implementations.

A knowledge of basic data processing techniques is necessary for the understanding of this document. In addition, the following manuals (referred to in this publication) provide information necessary for effective usage of MRX COBOL.

- **Program Library Services Reference** manual
- **Control Program and Data Management Services, Basic and Extended Reference** manuals

ACKNOWLEDGMENT

The following extract from **USA Standard COBOL, X3.23-1968,** is presented for the information and guidance of the user.

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

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

"No warranty, expressed or implied, is made by any contributor or by the COBOL Committee as to the accuracy and functioning of the 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 procedures for proposing changes should be directed to the Executive Committee of the Conference on Data Systems Languages.

"The authors and copyright holders of the copyrighted material used herein:

FLOW-MATIC (Trademark of Sperry Rand Corporation), Programming for the UNIVAC® I and II, Data Automation Systems Copyrighted 1958, 1959, by Sperry Rand Corporation; IBM Commercial Translator, Form No. F28-8013, copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell

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

FEATURES OF MRX COBOL

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

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

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

This document describes the MRX/40 and 50 OS COBOL (hereafter called MRX COBOL), which complies with the specifications of ANS COBOL and includes a number of Memorex extensions to it as well. The compiler supports the following standard levels of the processing modules defined in STANDARD COBOL.

- Low nucleus – defines the permissible character set and the basic elements of the language contained in each of the four COBOL divisions: Identification Division, Environment Division, Data Division, and Procedure Division.
- Medium table handling – allows the definition of tables and references to them through subscripts and indexes.
- Low sequential access – allows the records of a file to be read or written in a serial manner.
- Low random access – allows the records of a file to be read or written in a manner specified by the programmer.
- High segmentation – allows large programs to be split into segments that can be assigned to permanent or overlayable areas within a user's partition.
- Null sort – specifies that the COBOL Sort feature is not included in MRX COBOL.
- Null report writer – specifies that the COBOL Report Writer feature is not included in MRX COBOL.

- Low library – supports the retrieval of prewritten source program entries from a user library for inclusion in a COBOL program. The copy feature is not implemented as an integral part of the COBOL compiler; this feature is available to the user as a feature of the Operating System Librarian UPDATE utility.

- MRX COBOL Extensions

  1. Double or single quotes
  2. Linkage Section
  3. Comment lines (* or / in column 7)
  4. FILLER at the group level
  5. USAGE $\left\{\begin{array}{l}\text{COMP-3}\\\text{PACKED}\\\text{BINARY}\end{array}\right\}$
  6. INDEX-BLOCK SIZE clause
  7. Index File organization
  8. TO is optional on EQUAL TO relational operator

- Standard COBOL features not in MRX COBOL – Switches in Special-Names Section

- MRX COBOL features from higher levels of standard COBOL

  1. Nucleus

     a. Single digit level number

     b. Level numbers from 1 to 49

     c. ACCEPT . . . FROM

     d. DISPLAY . . . UPON

     e. Compare operands which are unequal in length (nonnumeric compare)

     f. Plural form of figurative constants

     g. Data-name may begin with numeric digit

     h. Symbolic Relational Operators

  2. Sequential and Random Access

     a. RESERVE ALTERNATE AREAS

     b. Data-name in VALUE clause of File Description (FD) entry

c. NO REWIND in OPEN

d. READ . . . INTO

e. WRITE . . . FROM

f. NO REWIND/LOCK in CLOSE

g. Data-name in AFTER ADVANCING clause

3. Table Handling

a. OCCURS . . . DEPENDING ON

TABLE OF CONTENTS

Section		Page
1	INTRODUCTION	1-1
	Input to COBOL Compiler	1-1
	Source Program	1-1
	Compiler Options	1-1
	Output from COBOL Compiler	1-3
	Object Program	1-3
	Listings	1-4
	COBOL Program Flow	1-5
2	FORMAT NOTATION	2-1
3	STRUCTURE OF COBOL	3-1
	Organization of the COBOL Program	3-1
	Structure of the COBOL Program	3-1
	COBOL Character Set	3-2
	Computer Characters	3-3
	Alphanumeric Characters	3-3
	Alphabetic Characters	3-3
	Numeric Characters	3-3
	Word Characters	3-3
	Punctuation Characters	3-3
	Special Characters	3-4
	Editing Characters	3-4
	Arithmetic Expression Characters	3-5
	Relation Condition Characters	3-5
	Character Strings	3-5
	Word	3-5
	Reserved Word	3-5
	Key Word	3-5
	Optional Word	3-5
	Connective	3-6
	Name	3-6
	Data-Name	3-6
	Identifier	3-6
	Procedure-Name	3-6
	File-Name	3-6
	Mnemonic-Name	3-6
	Index-Name	3-7
	System-Name	3-7
	Program-Name	3-7

TABLE OF CONTENTS (Continued)

Section		Page
	Constants	3-7
	Literals	3-7
	Numeric Literals	3-7
	Nonnumeric Literals	3-8
	Figurative Constants	3-8
	Picture Character String	3-9
	NOTE Character String	3-9
	Special Registers	3-9
4	USE OF COBOL CODING FORM	4-1
	Sequence Numbers	4-1
	Continuation of Lines	4-1
	Continuation of Nonnumeric Literals	4-1
	Continuation of Words and Numeric Literals	4-1
	Area A and Area B	4-1
	Division Header	4-3
	Section Header	4-3
	Paragraph-Names and Paragraphs	4-3
	Level Indicators and Level Numbers	4-3
	Blank Lines	4-3
	Comment Lines	4-4
5	IDENTIFICATION DIVISION	5-1
	Organization of the Identification Division	5-1
	Program-ID Paragraph	5-1
6	ENVIRONMENT DIVISION	6-1
	Organization of the Environment Division	6-1
	Configuration Section	6-1
	Source-Computer Paragraph	6-2
	Object-Computer Paragraph	6-2
	Special-Names Paragraph	6-3
	Maximum Configuration Section	6-4
	Input-Output Section	6-4
	File-Control Paragraph	6-4
	SELECT Clause	6-5
	ASSIGN Clause	6-5
	RESERVE Clause	6-6
	FILE-LIMIT Clause	6-7
	ACCESS MODE Clause	6-8
	PROCESSING MODE Clause	6-8
	ACTUAL KEY Clause	6-8

TABLE OF CONTENTS (Continued)

Section		Page
	Relative Files	6-8
	Indexed Files	6-10
	FORWARD KEY Clause	6-13
	INDEX-BLOCK Clause	6-13
	FILE-CONTROL Clause Restrictions	6-14
	Maximum FILE-CONTROL Paragraph	6-15
	I-O Control Paragraph	6-15
	RERUN Clause	6-15
	SAME AREA Clause	6-16
	Maximum I-O-CONTROL Paragraph	6-16
	Maximum Input-Output Section	6-17
	Maximum Environment Division	6-18
7	DATA DIVISION	7-1
	Organization of the Data Division	7-1
	Data Division Entries	7-2
	Concept of Levels	7-2
	Level Numbers	7-3
	Level Indicator	7-3
	File Section	7-3
	File Description Entry	7-3
	Record Description Entry	7-4
	Working-Storage Section	7-4
	Data Item Description Entries	7-4
	Record Description Entries	7-4
	Linkage Section	7-4
	File Description Entry — Details of Clauses	7-5
	File Description	7-5
	Block Contains Clause	7-5
	Record Contains Clause	7-6
	Data Records Clause	7-6
	Label Records Clause	7-7
	Value of Clause	7-7
	Maximum File Description Entry	7-8
	Data Description	7-8
	Data Description Entries	7-8
	Data Item Description Entries	7-9
	Record Description Entries	7-9
	Data Description Entry Clauses	7-9
	Level Number	7-9
	Data-Name or Filler Clause	7-10
	Redefines Clause	7-10
	Blank When Zero Clause	7-16
	Justified Clause	7-16

TABLE OF CONTENTS (Continued)

Section		Page
	Occurs Clause	7-17
	Picture Clause	7-17
	Symbols Used in the Picture Clause	7-17
	Repetition of Symbols	7-19
	Character String and Item Size	7-20
	Five Categories of Data	7-20
	Alphabetic Items	7-21
	Alphanumeric Items	7-21
	Numeric Items	7-21
	External Decimal	7-21
	Binary	7-23
	Internal Decimal	7-23
	Alphanumeric Edited Items	7-23
	Numeric Edited Items	7-24
	Three Classes of Data	7-24
	Editing Rules	7-25
	Simple Insertion Editing	7-25
	Special Insertion Editing	7-26
	Fixed Insertion Editing	7-26
	Floating Insertion Editing	7-27
	Zero Suppression and Replacement Editing	7-28
	Picture Clause Restrictions	7-29
	Synchronized Clause	7-31
	Slack Bytes	7-31
	Intra-Record Slack Bytes	7-32
	Inter-Record Slack Bytes	7-34
	Synchronized Clause Restrictions	7-34
	Usage Clause	7-34
	Value Clause	7-36
	Maximum Record or Data Item Description Entry	7-38
8	PROCEDURE DIVISION	8-1
	Organization of the Procedure Division	8-1
	Statements	8-2
	Compiler Directing Statements	8-2
	Conditional Statements	8-2
	Imperative Statements	8-2
	Sentences	8-3
	Conditions	8-4
	Test Conditions	8-4
	Class Condition	8-4
	Relation Condition	8-5
	Comparison of Numeric Operands	8-6
	Comparison of Nonnumeric Operands	8-6

TABLE OF CONTENTS (Continued)

Section	Page
Comparisons Involving Index-Names and/or Index Data Items	8-7
Permissible Comparisons of Subject and Object Operands	8-7
Conditional Statements	8-8
If Statement	8-9
Imperative Statements	8-10
Arithmetic Statements	8-10
Giving Option	8-11
Rounded Option	8-11
Size Error Option	8-11
Overlapping Operands	8-11
Add Statement	8-11
Subtract Statement	8-12
Multiply Statement	8-14
Divide Statement	8-16
Procedure Branching Statements	8-18
Go To Statement	8-18
Alter Statement	8-20
Perform Statement	8-21
Stop Statement	8-28
Exit Statement	8-28
Data Manipulation Statements	8-29
Move Statement	8-29
Examine Statement	8-33
Input-Output Statements	8-34
Open Statement	8-35
Seek Statement	8-36
Start Statement	8-36
Read Statement	8-37
Write Statement	8-39
Rewrite Statement	8-41
Delete Statement	8-42
Accept Statement	8-43
Display Statement	8-44
Close Statement	8-45
Subprogram Linkage Statements	8-48
Call Statement	8-48
Program Termination Considerations	8-51
Exit Program Statement	8-52
Stop Run Statement	8-52
Compiler-Directing Statements	8-52
Enter Statement	8-52
Note Statement	8-53

TABLE OF CONTENTS (Continued)

Section		Page
9	SPECIAL FEATURES	9-1
	Table Handling	9-1
	Table Definition	9-1
	References to Table-Items	9-3
	Subscripting	9-3
	Indexing	9-5
	Restrictions on Indexing and Subscripting	9-6
	Examples of Subscripting and Indexing	9-6
	Data Division Considerations for Table Handling	9-7
	Occurs Clause	9-8
	Usage Clause	9-10
	Procedure Division Considerations for Table Handling	9-11
	Relation Condition	9-11
	Set Statement	9-12
	Segmentation	9-13
	Organization	9-13
	Fixed Portion	9-13
	Independent Segments	9-13
	Segment Classification	9-14
	Segmentation Control	9-14
	Structure of Program Segments	9-14
	Segment-Limit	9-15
	Restrictions on Program Flow	9-15
	Alter Statement	9-15
	Perform Statement	9-15
	Source Program Library Facility	9-16
APPENDIX A	– GLOSSARY OF COBOL TERMS	A-1
APPENDIX B	– EBCDIC COLLATING SEQUENCE	B-1
APPENDIX C	– MRX COBOL RESERVED WORDS	C-1
APPENDIX D	– ANS STANDARD CONTROL CHARACTERS	D-1
APPENDIX E	– RECORDING MODES	E-1
APPENDIX F	– FILE PROCESSING SUMMARY	F-1
APPENDIX G	– INDEX – BLOCK SIZE FOR INDEXED FILES	G-1
APPENDIX H	– COBOL ERROR MESSAGES	H-1

LIST OF FIGURES

Figure		Page
1-1	COBOL Compiler Program Flow	1-6
6-1	Random Access of a Relative File	6-9
6-2	Sequential Access of an Indexed File	6-11
6-3	Random Access of an Indexed File	6-12
6-4	FILE-CONTROL Clause, Restrictions	6-14
7-1	Data Items Redefined Within an Area	7-12
7-2	Data Items Rearranged Within an Area	7-13
7-3	Internal Representation of Numeric Items	7-22
7-4	Class and Category of Elementary and Group Data Items	7-24
7-5	Examples of Simple Insertion Editing	7-25
7-6	Examples of Special Insertion Editing	7-26
7-7	Editing Sign Control Symbols and Results	7-27
7-8	Examples of Fixed Insertion Editing	7-27
7-9	Examples of Floating Insertion Editing	7-28
7-10	Zero Suppression and Replacement Editing	7-29
7-11	Insertion of Slack Bytes Between Occurrences	7-33
8-1	Allowable Forms of the Class Test	8-5
8-2	Relational Operators and Their Meanings	8-5
8-3	Examples of Comparisons of Numeric Operands	8-6
8-4	Examples of Comparisons of Nonnumeric Operands	8-7
8-5	ADD Statement Examples	8-13
8-6	SUBTRACT Statement Examples	8-15
8-7	MULTIPLY Statement Examples	8-17
8-8	DIVIDE Statement Examples	8-19
8-9	MOVE Statement Examples	8-32
8-10	Examples of Data Examination	8-34
8-11	CLOSE Option and File Type Comparison	8-47
8-12	Effect of Program Termination Statements Within Main Programs and Subprograms	8-51
9-1	Example of Table Indexing	9-7
9-2	Index-Names and Index Data Items – Permissible Comparisons	9-11

LIST OF TABLES

Table		Page
8-1	Permissible Comparisons of Subject and Object Operands	8-8
8-2	Permissible Moves	8-31

1. INTRODUCTION

COBOL is a programming language which is essentially machine independent. A program written in COBOL (source program) follows a set of formatting rules. This source program is input to the COBOL compiler where it is translated into a series of machine instructions which can be executed by the computer. The COBOL compiler is itself a program within the operating system. The output resulting from the compiler's translation is a relocatable object program.

INPUT TO COBOL COMPILER

Input to the COBOL compiler consists of the COBOL source program and the compiler options. The compiler options direct the compiler as to the type of output desired as well as certain input options.

SOURCE PROGRAM

The COBOL source program may be presented to the COBOL compiler by one of three means:

1. Directly from the card reader
2. From a spooled file on disc
3. From a user's source image library on disc (partitioned data set)

The user supplies the source program either directly with the compilation request in the form of cards, or indicates the source image library from which the source is to be read. When the source program is supplied by the card reader, the user has the option (using MRX/OS Control Language Services), prior to compilation, of spooling the source cards to disc. When a source image library is specified as the means of supplying the source program, the user must have previously placed the source program in the library.

COMPILER OPTIONS

These options will direct the compiler in its execution and specify the content and format of its output.

The options are supplied to the compiler via a //PAR control language statement, which is read by the compiler from the SYSIN file. The //PAR card keywords, as they relate to COBOL, and the resultant actions are listed in the following table.

//PAR STATEMENT Keywords

Keyword	Parameters	Default	Explanation
OBJECT=	YES NO	YES	OBJECT=YES or omission of keyword specifies output of relocatable object module. OBJECT=NO suppresses output.
IMEM=	input-name	none	An alphanumeric string of 1-8 characters specifying the cataloged name of the COBOL source program to be compiled. If this keyword is not specified, the source program is presumed to be on the card reader or input spool file.
OMEM=	output-name	PROGRAM-ID if OBJECT= YES	An alphanumeric string of 1-8 characters specifying the name under which the relocatable object program will be cataloged in the library. If OMEM is not specified and OBJECT=NO, a relocatable object module will not be produced. If OMEM is not specified and OBJECT= YES, the PROGRAM—ID is used for the catalog name of the relocatable object module.
RMARG=	nnn	SYSGEN value	A 2 to 3 digit number specifying the right margin column, 41-120, at which the compiler will stop processing the source input records.
SPACE=	nn	01	A 1 or 2 digit number specifying single or double spacing on the source listing. nn 1 or 01 single space 2 or 02 double space
MAXSIZ=	nnnnn	SYSGEN value	A 1 to 5 digit number specifying the approximate maximum number of cards in the source program.
LIST=	YES NO	YES	LIST=NO suppresses source program listing. LIST=YES or omission of keyword supplies source program listing.
ERROR=	YES NO	YES	ERROR=YES or omission of keyword supplies error message listing of warning errors. ERROR=NO suppresses listing. Fatal errors are listed regardless of option.

Keyword	Parameters	Default	Explanation
XREF=	YES NO	NO	XREF=YES specifies printing a cross-reference list. XREF=NO or omission of keyword suppresses listing.
DMAP=	YES NO	NO	DMAP=YES specifies printing Data Map. DMAP=NO or omission of keyword suppresses map.
PMAP=	YES NO	NO	PMAP=YES specifies printing procedure map. PMAP=NO or omission of keyword suppresses map.
SUBCK=	YES NO	NO	SUBCK=YES specifies generation of object code to confirm that the resolved subscript value does not exceed the number of entries in the associated tables. SUBCK=NO or omission of keyword suppresses generation of object code.
DATACK=	YES NO	NO	DATACK=YES specifies generation of object code to check that only numeric digits are contained in external decimal and packed decimal fields used in the IF and arithmetic verbs.
QUOTE=	QUOTE APOST	APOST	This parameter specifies whether the double quotation mark, QUOTE, or the apostrophe, APOST, is to be used as the quote character during compilation.

OUTPUT FROM COBOL COMPILER

The compiler output consists of the relocatable object program and its listings. The object program output is optional; it may be suppressed by a keyword parameter (compiler option) on the //PAR card. A summary listing will always be produced. Other categories of listings may be selected or suppressed from the output by keyword parameters on the //PAR card.

OBJECT PROGRAM

The object program output will be sent to a user library with the member name (by which the program is cataloged on the library), supplied by the user in the compilation options.

LISTINGS

With the exception of a program summary listing which is always produced, the entries in the listing are optional as selected by keyword parameters on the //PAR card. The format and content of each is as follows:

- Summary – consists of general compiler statistics and a memory map layout.

  The general compiler statistics are: date of compilation, number of fatal errors, and options used in compilation. These are self-explanatory.

  The memory map layout provides the user with a convenient index of his memory dump. Its contents and format are as follows:

  ```
  ENTRY POINT LIST   :   NAME
  MEMORY LAYOUT      :   NAME     Relative address
                          .              .
                          .              .
                          .              .
  EXTERNALS LIST     :   NAME
                          .
                          .
                          .
  ```

- Source listing – consists of source image and generated line numbers. An S in print position 1 denotes a sequence error in columns 1-6 of the source card.

- Data map – consists of definition name, line number and relative address, in the order of appearance in the source program.

- Cross reference list – consists of file name, index name, data name, and procedure name lists. Each list is in ascending alphabetical order with its reference line numbers in ascending numerical order. In addition, the relative address of each name (excluding procedure names) is given.

- Procedure map – consists of the entire instruction set generated by each source line. The line number is printed once at the beginning of each set.

- Error listing – consists of the following information:

  1. Line number where the error occurred

  2. F, U, or W indicating fatal, ANS, or warning errors respectively

3. Error identification number

4. Clause — clause first code

5. Short but comprehensive explanation of the error code

6. A total count of each type of error (F, U and W)

COBOL PROGRAM FLOW

Figure 1-1 summarizes the flow of a COBOL program from the time the compiler receives it to the time it is sent out in some form.

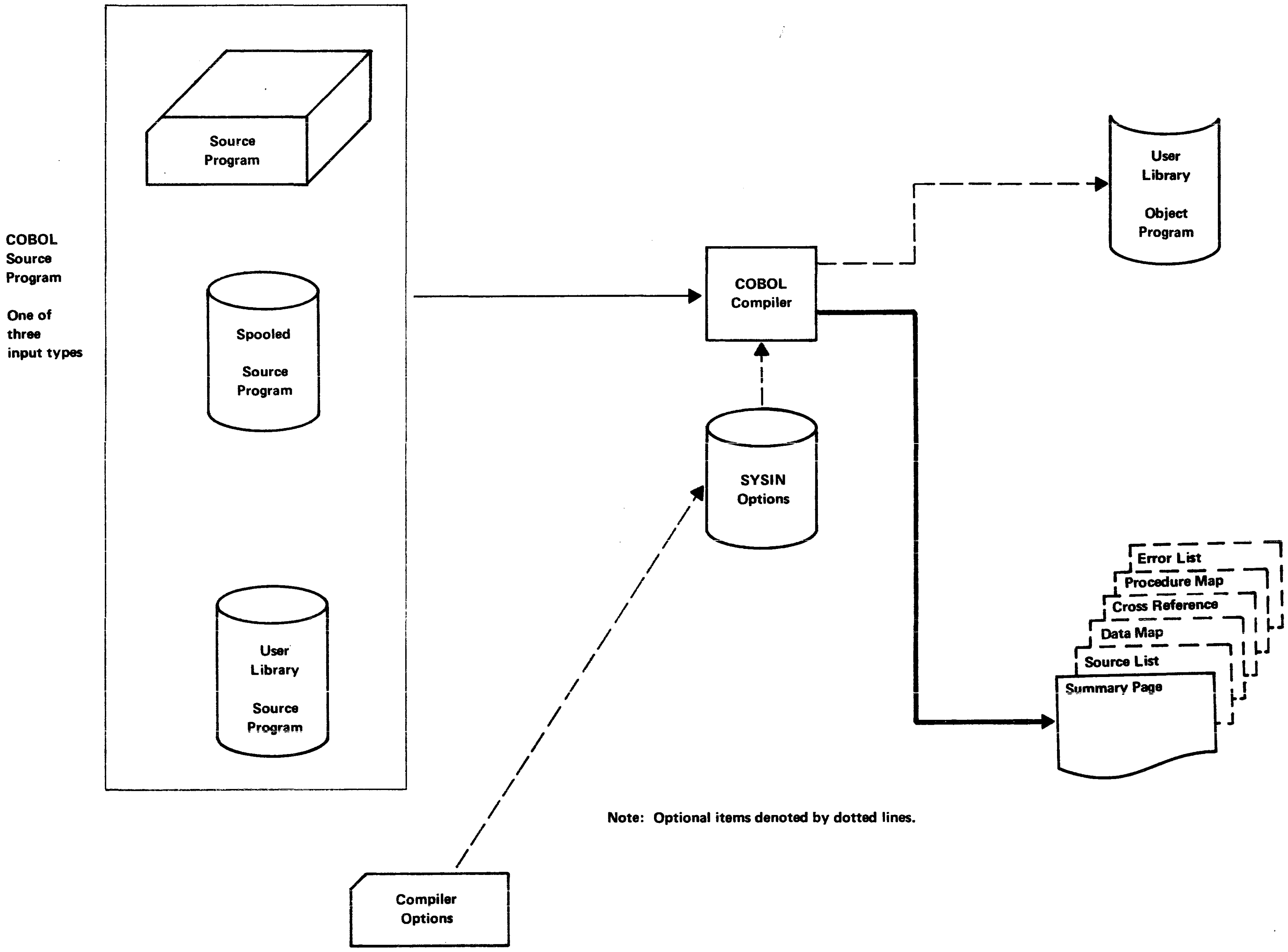


Figure 1-1. COBOL Compiler Program Flow

2. FORMAT NOTATION

Throughout this publication, the format of COBOL statements is presented in a uniform system of notation. The following system of notation is used to describe the format of COBOL statements.

- Upper case characters that are underlined are reserved key words and are required.
- Upper case characters that are not underlined are reserved optional words. They may be used for the sake of readability.
- Lower case characters represent information supplied by the programmer.
- Square brackets [ ] indicate that the contents enclosed are optional and may be included in the source program as necessary.
- Braces { } indicates that a selection of one of the options contained within must be made.
- Ellipsis . . . indicates that the preceding unit may occur once or any number of times in succession.
- Plus sign (+) and minus sign (-) when appearing in formats, although not underlined, are required when such formats are used.
- Punctuation and special characters (with the exception of the previously mentioned) are required where shown.
- A period or comma, when used, must not be preceded by a space but must be followed by a space.
- A left parenthesis must not be followed immediately by a space; a right parenthesis must not be preceded immediately by a space.
- At least one space must appear between two successive words or literals. Two or more successive spaces are treated as a single space, except within nonnumeric literals.
- An arithmetic operator must always be preceded by a space and followed by a space.
- Extensions to ANS COBOL and all references to such extensions have a shaded background.

3. STRUCTURE OF COBOL

COBOL is a structured language. The programmer must write his individual problem program within a framework of words that have a particular meaning to the COBOL compiler. The result is the performance of a standard action on specific units of data.

ORGANIZATION OF THE COBOL PROGRAM

A COBOL source program consists of four major divisions. Each is identified by a division header in the proper order and sequence as shown in the following paragraphs:

1. IDENTIFICATION DIVISION – names the program
2. ENVIRONMENT DIVISION -- specifies equipment configuration and I/O media.
3. DATA DIVISION – defines the characteristics of data to be processed by the object program.
4. PROCEDURE DIVISION – describes the procedure used in manipulating the data.

NOTE

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

STRUCTURE OF THE COBOL PROGRAM

The structure of a basic COBOL program would appear as follows:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.program-name.
[AUTHOR.[comment-entry]...]
[INSTALLATION.[comment-entry]...]
[DATE WRITTEN.[comment-entry]...]
[SECURITY.[comment-entry]...]
[REMARKS.[comment-entry]...]

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

```
DATA DIVISION.
[FILE SECTION.
 {file description entry
 {record description entry} . . .} . . .]
[WORKING-STORAGE SECTION.
[data item description entry] . . .
[record description entry] . . .]
[LINKAGE SECTION.
[data item description entry] . . .
[record description entry] . . .]

PROCEDURE DIVISION [USING identifier-1 [identifier-2] . . .].
[section-name SECTION [priority] .]
{ paragraph-name . {sentence} . . . } . . .
END-PROGRAM.
[END-COMPILATION.]
```

COBOL CHARACTER SET

The basic indivisible unit of the COBOL Language is the character. The complete character set for MRX COBOL consists of the following 47 characters.

Character	Meaning
0 – 9	Digit
A – Z	Letter
	Space
+	Plus sign
-	Minus sign (or hyphen)
*	Asterisk
/	Stroke (virgule, slash)
$	Currency sign
,	Comma
.	Period
" or '	Apostrophe (quotation mark)
(	Left parenthesis
)	Right parenthesis
>	Greater than
<	Less than
=	Equals

Characters are further classified in an array of subsets. A character may be defined as being part of one or more of the following listed subsets.

COMPUTER CHARACTERS

A computer character is a character that belongs to the Extended Binary Coded Decimal Interchange Code (EBCDIC) set.

ALPHANUMERIC CHARACTERS

An alphanumeric character is any character in the computer's character set.

ALPHABETIC CHARACTERS

An alphabetic character is a character that belongs to the following set of letters: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z, and the space.

NUMERIC CHARACTERS

A numeric character is a character that belongs to the following set of digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9.

WORD CHARACTERS

The characters used in words in a COBOL source program are as follows:

- 0 through 9
- A through Z
- - (hyphen)

PUNCTUATION CHARACTERS

A punctuation character is a character belonging to the following set.

Character	Meaning
	Space
,	Comma
.	Period
" or '	Quotation mark
(	Left parenthesis
)	Right parenthesis

SPECIAL CHARACTERS

A special character is a character that belongs to the following set.

Character	Meaning
+	Plus sign
-	Minus sign
*	Asterisk
/	Stroke (virgule, slash)
$	Currency sign
,	Comma (decimal point)
.	Period (decimal point)
" or '	Quotation mark
(	Left parenthesis
)	Right parenthesis
<	Less than
>	Greater than
=	Equals

EDITING CHARACTERS

An editing character is a single character or a fixed two-character combination belonging to the following set.

Character	Meaning
B	Space
0	Zero
+	Plus
-	Minus
CR	Credit
DB	Debit
Z	Zero suppression
*	Check protection
$	Currency sign
,	Comma (decimal point)
.	Period (decimal point)

ARITHMETIC EXPRESSION CHARACTERS

MRX COBOL does not implement the use of a minus sign (-) preceding a variable or left parenthesis within an arithmetic expression. Arithmetic expressions are limited to those expressions used to specify relative indexing. The characters used are as follows:

Character	Meaning
+	Addition
-	Subtraction

RELATION CONDITION CHARACTERS

Relational operators can be used in place of the relational characters.

Examples: GREATER, LESS, EQUAL TO

CHARACTER STRINGS

A character string is a set of contiguous characters which form a word, a name, a constant, a PICTURE in the Data Division, or a NOTE in the Procedure Division.

It is delimited by a space, a period, a comma, or a right parenthesis.

WORD

A sequence of not more than 30 characters chosen from the word character set. A word may not begin or end with a hyphen.

RESERVED WORD

A word that has a preassigned meaning to the COBOL compiler. It may not appear as a user defined word unless it is a nonnumeric literal enclosed by quotation marks.

KEY WORD

A word that is required when it appears in a COBOL entry.

OPTIONAL WORD

An optional word that may appear at the user's discretion for the sake of readability. Misspelling of an optional word or its replacement by another word is not allowed.

CONNECTIVE

A comma used to link two or more subscripts or index expressions in a subscript data name reference. This is the only connective included in MRX COBOL.

NAME

There are seven types of names used in a MRX COBOL program: data-names, procedure-names, file-names, mnemonic-names, index-names, system-names and program-names.

DATA-NAME

A word that contains at least one alphabetic character (not necessarily the first). It names an entry in the Data Division. All data names must be unique as qualification is not included in MRX COBOL.

IDENTIFIER

An identifier is a data-name, followed, as required, by the syntactically correct combination of subscripts or indexes necessary to make unique reference to a data item.

PROCEDURE-NAME

May be a paragraph name or a section name used to refer to that paragraph or section in the source program. It may be composed of solely numeric characters. If so, data names are equivalent only if they are composed of the same number of digits and have the same value. No name can be both a data-name and a procedure-name.

FILE-NAME

A file-name is a word with at least one alphabetic character that names a file described in the Data Division. It is formed according to the rules for formation of a data-name.

MNEMONIC-NAME

A mnemonic-name is a word, supplied by the programmer, that is associated in the Environment Division with a specific implementor-name. An implementor-name is a reserved word that refers to a particular feature available on a Memorex computer system. Mnemonic-names are formed according to the rules for formation of a data-name.

INDEX-NAME

An index-name is a word with at least one alphabetic character that names an index associated with a specific table. It is formed according to the rules for formation of a data-name.

SYSTEM-NAME

A system-name is a word that specifies the external name of a file, a device class, and an organization method. The external name consists of from one to eight alphanumeric characters. The first character must be alphabetic.

PROGRAM-NAME

A program-name is a word that identifies a COBOL source program. The program-name consists of alphanumeric characters, the first of which must be alphabetic.

CONSTANTS

A constant is a unit of data whose value is not subject to change. The two types of constants are:

- Literals
- Figurative constants

LITERALS

A literal is a string of characters whose value is implied by the ordered set of characters of which the literal is composed. Every literal belongs to one of two types: numeric or nonnumeric.

Numeric Literals

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

The rules for formation of a numeric literal are as follows:

- It must contain from 1 to 18 digits.
- It must not contain more than one sign character. If a sign is used, it must appear as the leftmost character of the literal. If the literal is unsigned, the literal is positive.
- It must not contain more than one decimal point. The decimal point is treated as an assumed decimal point and may appear anywhere within the literal except as the rightmost character. If the literal contains no decimal point, it is an integer.

The value of a numeric literal is the algebraic quantity represented by the characters in the numeric literal.

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

Nonnumeric Literals

A nonnumeric literal is a string of 1 to 120 alphanumeric characters bounded by quotation marks. Any character in the alphanumeric character set may be included in the literal with the exception of the quotation mark, which has the special purpose of enclosing the character string.

The value of a nonnumeric literal is the string of characters itself, excluding the quotation marks. Any spaces enclosed in the quotation marks are part of the nonnumeric literal and, therefore, are part of the value.

FIGURATIVE CONSTANTS

A figurative constant is a reserved word that represents a numeric value, a character, or a string of characters. Such words must not be enclosed in quotation marks when used as figurative constants.

A figurative constant can be used wherever a literal appears in a format. However, when the literal is restricted to numeric characters only, all figurative constants except zero are illegal.

The figurative constants and their meanings are as follows:

Figurative Constant	Meaning
ZERO ZEROS ZEROES	Represents the value 0, or one or more occurrences of the character 0, depending on context.
SPACE SPACES	Represents one or more blanks or spaces.
HIGH-VALUE HIGH-VALUES	Represents one or more occurrences of the character that has the highest value in the computer's collating sequence. The character for HIGH-VALUE is the hexadecimal FF.
LOW-VALUE LOW-VALUES	Represents one or more occurrences of the character that has the lowest value in the computer's collating sequence. The character for LOW-VALUE is the hexadecimal 00.
QUOTE QUOTES	Represents one or more occurrences of the quotation mark character. The word QUOTE cannot be used in place of a quotation mark to enclose a nonnumeric literal.

PICTURE CHARACTER STRING

A PICTURE character string consists of certain combinations of characters in the COBOL character set used as symbols. The allowable combinations are explained under the *PICTURE Clause* in Section 7.

NOTE CHARACTER STRING

A NOTE character string may consist of any combination of the characters from the computer's character set. NOTE is described under *NOTE Statement* in Section 8.

SPECIAL REGISTERS

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

The word TALLY is the name of a special register whose implicit description is that of an integer of five digits without an operational sign and whose implicit USAGE is COMPUTATIONAL.

The primary use of the TALLY register is to hold information produced by the EXAMINE statement. References to TALLY may appear wherever an elementary data item of integral value may appear (refer to the *EXAMINE Statement* in Section 8).

4. USE OF COBOL CODING FORM

The reference format provides a standard method for writing COBOL source programs. The format is described in terms of character positions in a line on an I/O medium. Punched cards are the initial input medium to the COBOL compiler. The compiler accepts source programs written in reference format (Figure 4-1) and produces an output listing of the source program in the same reference format.

SEQUENCE NUMBERS

A sequence number, consisting of six digits in the sequence area, is used to numerically identify each card image to be compiled by the COBOL compiler. The use of sequence numbers is optional, but if present they must be in ascending order. A card out of sequence will be flagged with an S preceding the sequence number in the source listing output. A card with a blank sequence number is not checked for sequence purposes.

CONTINUATION OF LINES

Any sentence or entry that requires more than one line is continued by starting subsequent lines in Area B (which starts in column 12). These subsequent lines are called continuation lines. The line being continued is called the continued line. If the sentence or entry occupies more than two lines, all lines other than the first and last are both continuation and continued lines.

CONTINUATION OF NONNUMERIC LITERALS

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

CONTINUATION OF WORDS AND NUMERIC LITERALS

A word or numeric literal cannot be broken in such a way that part of it appears on a continuation line.

AREA A AND AREA B

Area A (columns 8 through 11) is reserved for the beginning of division headers, section-names, paragraph-names, level indicators, and certain level numbers. Area B occupies columns 12 through the right margin (RMARG) and is used for statements and sentences of the main COBOL program.

MEMOREX

COBOL Coding Form

Punching Instructions

Graphic						
Punch						

Date________ Page____ of____

Programmer________

Program________

SEQUENCE (PAGE)	SEQUENCE (SERIAL)	CONT.	A	B COBOL STATEMENT	IDENTIFICATION
1 2 3	4 5 6	7	8 9 10 11	12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72	73 74 75 76 77 78 79 80
1 2 3	4 5 6	7	8 9 10 11	12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72	73 74 75 76 77 78 79 80

DIVISION HEADER

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

SECTION HEADER

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

PARAGRAPH-NAMES AND PARAGRAPHS

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

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

LEVEL INDICATORS AND LEVEL NUMBERS

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

In those data description entries that begin with a level number 1 or 77, the level number begins in Area A followed in Area B by its associated data-name and appropriate descriptive information.

In those data description entries that begin with level numbers 2 through 49, the level number may begin anywhere in Area A or Area B, followed in Area B by its associated data-name and descriptive information.

BLANK LINES

A blank line is one that contains nothing but spaces from column 7 through the right margin (RMARG) inclusively. A blank line may appear anywhere in the source program, except immediately preceding a continuation line.

COMMENT LINES

Explanatory comments may be inserted on any line within a source program by placing an asterisk (*) or a stroke (/) in the continuation area of the line (column 7).

Any combination of the characters from the EBCDIC set may be included in Area A and B of that line. The complete line will be on the source listing but serve no other purpose than that of documentation.

When the comment indicator is a stroke (/), a page eject is performed prior to printing the comment.

5. IDENTIFICATION DIVISION

The Identification Division is the first division and must be included in every COBOL source program. The Identification Division assigns a name to the source program, the resultant output listing, and possibly the object program. In addition, the user may include the date the program is written, the author of the program, and other such information as desired, described in the following paragraphs.

ORGANIZATION OF THE IDENTIFICATION DIVISION

The Identification Division must begin with the reserved words IDENTIFICATION DIVISION followed by a period and a space. Each comment-entry may be any combination of the characters from the EBCDIC set, organized to conform to sentence and paragraph structure.

Fixed paragraph-names identify the type of information contained in the paragraph. The name of the program must be given in the first paragraph, which is the PROGRAM-ID paragraph. The other paragraphs are optional; they may be included in this division at the user's discretion, in order of presentation shown by the following format:

```
IDENTIFICATION DIVISION.

PROGRAM-ID. program-name.

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

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

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

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

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

PROGRAM-ID PARAGRAPH

The following text defines the PROGRAM-ID paragraph. While the other paragraphs are not defined, each general format is formed in the same manner.

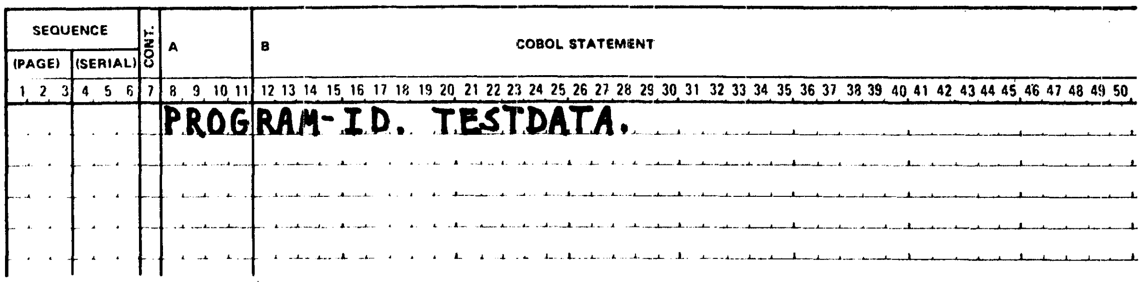
The PROGRAM-ID paragraph gives the name by which a program is identified. Its format is:

```
PROGRAM-ID. program-name.
```

The program-name must begin with an alphabetic character followed by up to 29 alphanumeric or hyphen (-) characters. Only the first six characters of program-name are used as the identifying name of the program. The use of a hyphen within these six character positions is illegal.

The PROGRAM-ID paragraph must contain the name of the program and must be present in every program. The program-name identifies the source program, all listings pertaining to a particular program, and possibly the object program.

Following is an example of a PROGRAM-ID paragraph:



SEQUENCE		CONT.	A	B COBOL STATEMENT
(PAGE)	(SERIAL)			
1 2 3	4 5 6	7	8 9 10 11	12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
			PROG	RAM-ID. TESTDATA.

Note that only the first six characters, TESTDA, are used to identify the program.

6. ENVIRONMENT DIVISION

The Environment Division specifies a standard method of expressing those aspects of a data processing problem that are dependent upon the physical characteristics of a specific computer. This division allows specification of the configuration of the compiling computer and the object computer. In addition, information relating to input-output control, special hardware characteristics, and control techniques can be given.

The Environment Division must be included in every COBOL source program as the second division.

ORGANIZATION OF THE ENVIRONMENT DIVISION

The Environment Division must begin with the reserved words ENVIRONMENT DIVISION followed by a period and a space.

Two sections make up the Environment Division: the Configuration Section and the Input-Output Section. The following is a general outline of the sections and paragraphs in the Environment Division. The order of presentation in the source program is also defined.

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

CONFIGURATION SECTION

The Configuration Section begins with the reserved words CONFIGURATION SECTION followed by a period and a space.

This section describes the characteristics of the source computer and the object computer, and is divided into three paragraphs: the SOURCE-COMPUTER paragraph, the OBJECT-COMPUTER paragraph, and the SPECIAL-NAMES paragraph.

SOURCE-COMPUTER PARAGRAPH

The SOURCE-COMPUTER paragraph identifies the computer on which the source program is to be compiled. Its format is:

SOURCE-COMPUTER. computer-name.

Computer-name must conform to the rules for formation of a data-name.

The SOURCE-COMPUTER paragraph serves for documentation purposes only. An example of a SOURCE-COMPUTER paragraph is as follows:

SEQUENCE (PAGE) 1 2 3	SEQUENCE (SERIAL) 4 5 6	CONT. 7	A 8 9 10 11	B 12 … 50 COBOL STATEMENT
			SOURCE-COMPUTER. MRX-4Ø.	

OBJECT-COMPUTER PARAGRAPH

The OBJECT-COMPUTER paragraph identifies the computer on which the program is to be executed. Its format is:

```
OBJECT-COMPUTER. computer-name

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

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

Computer-name must conform to the rules for formation of a data-name.

If the configuration implied by computer-name comprises more or less equipment than is actually needed by the object program, the descriptive clauses following computer-name permit the specification of the required configuration.

With the exception of the SEGMENT-LIMIT clause (described in Section 9), the OBJECT-COMPUTER paragraph serves for documentation purposes only. An example of an OBJECT-COMPUTER paragraph is as follows:

SEQUENCE (PAGE) 1 2 3	SEQUENCE (SERIAL) 4 5 6	CONT. 7	A 8 9 10 11	B 12 … 50 COBOL STATEMENT
			OBJECT-COMPUTER. MRX-4Ø.	

SPECIAL-NAMES PARAGRAPH

The SPECIAL-NAMES paragraph provides a means of relating implementor-names to user-specified mnemonic-names. Its format is:

```
SPECIAL-NAMES  [implementor-name IS mnemonic-name] . . .
     [CURRENCY SIGN IS literal]
     [DECIMAL-POINT IS COMMA] .
```

The SPECIAL-NAMES paragraph is required if mnemonic-names, the DECIMAL-POINT clause, or the CURRENCY SIGN clause are used. Otherwise, the paragraph is optional. If the paragraph is specified, it must appear in the order shown.

Implementor-name may be chosen from the following list:

- SYSIN
- SYSOUT
- CONSOLE

The literal which appears in the CURRENCY SIGN IS literal clause is used in the PICTURE clause to represent the currency symbol. The literal is limited to a single character, but must *not* be one of the following:

- Digits 0 through 9
- Alphabetic characters A, B, C, D, P, R, S, V, X, Z, or the space
- Special characters * + - , . ( ) ' < > =

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

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

An example of a SPECIAL-NAMES paragraph is as follows:

SEQUENCE (PAGE) 1 2 3	(SERIAL) 4 5 6	CONT. 7	A 8 9 10 11	B 12 … 50 COBOL STATEMENT
			SPEC	IAL-NAMES.
				SYSIN IS READER
				CURRENCY SIGN IS 'Y'
				DECIMAL POINT IS COMMA.

MAXIMUM CONFIGURATION SECTION

If the user chooses to specify all options are available in the Configuration Section, the general format would appear as follows:

```
CONFIGURATION SECTION.

SOURCE-COMPUTER.  computer-name.

OBJECT-COMPUTER.  computer-name.

  ┌                              ⎧ WORDS      ⎫ ┐
  │ MEMORY SIZE integer          ⎨ CHARACTERS ⎬ │
  └                              ⎩ MODULES    ⎭ ┘

   [SEGMENT-LIMIT IS priority-number].

[SPECIAL-NAMES.  [implementor-name IS mnemonic name]...

   [CURRENCY SIGN IS literal]

   [DECIMAL POINT IS COMMA].]
```

INPUT-OUTPUT SECTION

The Input-Output Section begins with the reserved words INPUT-OUTPUT SECTION followed by a period and a space.

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 and the I-O-CONTROL paragraph.

FILE-CONTROL PARAGRAPH

The FILE-CONTROL paragraph names each file, identifies the file medium, and allows particular hardware assignments. The general format is as follows:

```
FILE-CONTROL.

    SELECT Clause

    ASSIGN Clause

    [RESERVE Clause]

    [FILE-LIMIT Clause]

    [ACCESS MODE Clause]

    [PROCESSING MODE Clause]

    [ACTUAL KEY Clause]

    [FORWARD KEY Clause]

    [INDEX-BLOCK Clause] . ...
```

The FILE-CONTROL paragraph begins with the reserved word FILE-CONTROL followed by a period and a space. The clauses must appear in the order shown.

SELECT Clause

The SELECT clause is used to name each file in a program. The format is as follows:

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

Each file-name described in the Data Division must be named once and only once in the FILE-CONTROL paragraph following the key word SELECT. Each selected file must have a file description entry in the Data Division.

ASSIGN Clause

The ASSIGN clause is used to assign a file to an external medium. The format is as follows:

$$\underline{\text{ASSIGN}}\ \text{TO}\ [\text{integer}]\ \text{system-name-1}\ [\text{system-name-2}]\ldots$$
$$\left[\ \underline{\text{FOR MULTIPLE}}\quad \left\{\begin{array}{l}\underline{\text{REEL}}\\ \underline{\text{UNIT}}\end{array}\right\}\right]$$

Integer indicates the number of input-output units of a given number to be assigned to the file-name. The compiler, however, determines the number of units to be assigned, so the integer option has the function of a comment.

System-name specifies the external name of a file, a device class, and an organization method. Only system-name-1 is processed. All other system-names, if present, are treated as comments.

System-name has the following structure:

name[-organization] [-class]

Name consists of from one to eight alphanumeric characters, the first of which must be alphabetic, and represents the external name of the file. It is the name specified as the file identifier on the //DEFINE statement (described in the **MRX/OS Control Language Services, Extended Reference** manual).

Organization is a one-character field that specifies the file organization. The file organization codes are as follows:

- S — sequential files
- R — relative files
- I — indexed files

If organization is not specified, a sequential organization is assumed.

Class is a one-character field that represents the device class. The class codes are:

- D — disc devices
- T — tape devices
- U — unit record devices

If class is not specified, a disc device is assumed.

The FOR MULTIPLE REEL/UNIT clause is applicable whenever the number of tape units or mass storage devices assigned might be less than the number of reels or units in the file. The system, however, will automatically handle volume switching for sequentially accessed files, giving this clause the function of a comment if specified. All volumes must be mounted for randomly accessed files.

An example of an ASSIGN clause is as follows:

SEQUENCE		CONT.	A	B COBOL STATEMENT
(PAGE)	(SERIAL)			
1 2 3	4 5 6	7	8 9 10 11	12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
				SELECT FILE-IN ASSIGN TO SYSØ14-R-D.

RESERVE Clause

The RESERVE clause allows the user to modify the number of input-output areas allocated by the compiler. The format is as follows:

$$\underline{\text{RESERVE}} \begin{Bmatrix}\text{integer}\\ \underline{\text{NO}}\end{Bmatrix} \text{ ALTERNATE } \begin{Bmatrix}\text{AREA}\\ \text{AREAS}\end{Bmatrix}$$

A minimum of one buffer is required for a file. The ALTERNATE AREAS option reserves an addition area for the file in addition to the original area. Integer must be unsigned and have a value of 1. Therefore, if this clause is specified, one additional buffer may be assigned.

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

The RESERVE clause may be specified only for a sequential or relative file that is accessed in sequential mode.

An example of a RESERVE clause is as follows:

SEQUENCE		CONT.	A	B	COBOL STATEMENT
(PAGE)	(SERIAL)				
1 2 3	4 5 6	7	8 9 10 11	12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50	
				RESERVE 1 ALTERNATE AREA	

FILE-LIMIT Clause

The FILE-LIMIT clause specifies the address range of a mass storage file. The format is as follows:

$$\left\{\begin{array}{l}\underline{\text{FILE-LIMIT IS}}\\ \underline{\text{FILE-LIMITS ARE}}\end{array}\right\}\ \text{integer-1}\ \underline{\text{THRU}}\ \text{integer-2}$$

Integer-1 represents the logical beginning of the mass storage file. Integer-2 represents the logical end of the mass storage file. Neither integer-1 nor integer-2 may exceed the value of 2^{32}-1. Integer-2 may not be less than integer-1. The value of integer-1 must be greater than zero.

For a relative file processed in random access mode, the FILE-LIMIT clause specifies that logical records are obtained or placed randomly in the mass storage file within the limits specified. The contents of the ACTUAL KEY data items that are not within these limits cause the execution of the INVALID KEY clause on READ, WRITE, and SEEK statements.

For a relative file processed sequentially integer-1 specifies the first record to be read or written. Sequential processing of records continues through integer-2. If the file limit clause is omitted processing begins at the first record and continues in sequence till EOG in the case of read, or end of reserved area in the case of write.

If specified for a sequential or indexed file, the clause is treated as a comment.

When a file is initially created a permanent bias is set by Data Management. The bias set is the value of integer minus 1. In subsequent references to the file, an actual key set by the programmer specifies the desired record. The bias will be subtracted from this actual key (by the operating system) giving the relative record position on the file, and the operating system will position to that record.

The limits specified by the value of integer-2, less the value of integer minus 1 must be within the limits of the allocated space. At later accesses to the file, it is not required that the file-limits be equivalent to the file-limits specified at creation time. It is required, however, that the lower limit specified is not less than the permanent bias of the file.

ACCESS MODE Clause

The ACCESS MODE clause defines the manner in which records of a file are to be accessed. The format is as follows:

$$\underline{\text{ACCESS}}\ \text{MODE}\ \underline{\text{IS}} \left\{ \begin{array}{l} \underline{\text{SEQUENTIAL}} \\ \underline{\text{RANDOM}} \end{array} \right\}$$

If this clause is not specified, sequential access is assumed.

If ACCESS IS SEQUENTIAL, records are placed or obtained sequentially. That is, the next logical record is made available from the file when the READ statement is executed, or the next logical record is placed into the file when a WRITE statement is executed. ACCESS IS SEQUENTIAL may be applied to files assigned to tape, unit-record, or mass storage devices.

For ACCESS IS RANDOM, storage and retrieval are based on an actual key associated with each record (refer to *ACTUAL KEY Clause* in following text). When the RANDOM option is specified, the file must be assigned to a mass storage device. ACCESS IS RANDOM may be specified when file organization is relative or indexed.

PROCESSING MODE Clause

This clause is used to indicate that the logical processing is sequential. The entry is optional; it serves the function of documentation only. The format is as follows:

$$\underline{\text{PROCESSING}}\ \text{MODE IS}\ \underline{\text{SEQUENTIAL}}$$

ACTUAL KEY Clause

An ACTUAL KEY clause specifies a key that is used by the system to locate a logical record in a relative or indexed file. The format is as follows:

$$\underline{\text{ACTUAL}}\ \text{KEY}\ \underline{\text{IS}}\ \text{data-name}$$

Relative Files

The ACTUAL KEY clause is required for a relative file only when it is accessed randomly.

Data-name must be an unsigned integer numeric item defined in the File, Working-Storage, or Linkage section. Therefore, data-name must be an elementary item. The numeric value of data-name must not exceed 2^{31}-1, or left truncation will occur on the significant digits.

The contents of data-name is used to locate a specific relative position within the file when a SEEK statement is executed, or if no SEEK statement is executed, when a READ or a WRITE statement is executed. The relative position located corresponds directly to the numeric value contained in data-name less the value of the file-bias recorded at generation time.

If the value contained in data-name is not within the limit specified by the FILE-LIMITS clause, the imperative statement following INVALID KEY will be executed. (INVALID KEY is described in Procedure Section.)

Example:

At file creation time, the lower file limit is set to 1; the upper limit is set to 300. These limits allow the user to specify up to 300 records.

The lower file limit minus 1 becomes the permanent file bias and is used in conjunction with the actual key to locate a particular record within a file.

Figure 6-1 shows how the actual key is used to locate a record during a read operation.

Steps to Locate a Record

Step 1. The user sets the actual key.

ACTUAL KEY IS DN2

where DN2 = 200

Step 2. Operating system computes the relative record position of the record to be read.

ACTUAL KEY	200
-File bias (1 - 1 = 0)	-0
Relative record position	200

Step 3. Operating system retrieves relative record 200.

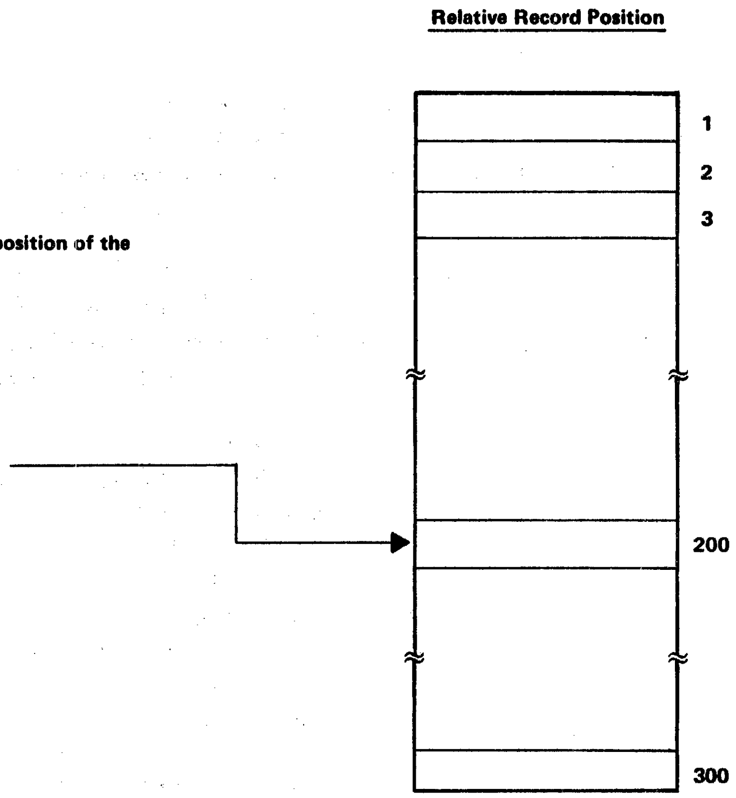


Figure 6-1. Random Access of a Relative File

Indexed Files

The ACTUAL KEY clause is required for an indexed file in the following instances:

- The file is accessed randomly
- The file is created
- The file is updated in sequential access mode

The data-name used in the ACTUAL KEY clause contains a symbolic value which identifies a specific record within the file. Data-name may be defined in the File, Working-Storage, or Linkage Section.

The total length of the actual key must not exceed 100 bytes.

The following discussion specifies when the ACTUAL KEY clause is required depending on the access mode and the processing operation of the file.

1. *Indexed file processed in sequential access mode.* When an indexed file is accessed sequentially, all records are accessed in the order of the primary key values. Keys are sequenced according to their position in the COBOL collating sequence.

   a. Sequential retrieval of records in an indexed file is performed by a READ statement. The record retrieval is the next in the logical sequence of the primary key. The ACTUAL KEY clause is not required for sequential retrieval.

   b. Sequential update of records in an indexed file is performed by a REWRITE statement. The execution of the REWRITE statement must be logically preceded by the execution of a READ statement. The ACTUAL KEY clause is required for sequential update of records. The value of the ACTUAL KEY must correspond to the primary key value of the record obtained in the preceding READ. If the two key values are not equal, the imperative statement following INVALID KEY will be executed.

   c. Sequential addition of records in an indexed file is performed by the WRITE statement. Specification of an ACTUAL KEY clause is required for sequential addition of records. The value of the actual key must be greater than the key value of the last record read, and less than the key value of the next record in sequence. If the value of the actual key is outside this range, the imperative statement following INVALID KEY will be executed.

d. Sequential creation of records in an indexed file is performed by the WRITE statement. Specification of an ACTUAL KEY clause is required for sequential creation of records. The value of the actual key must be unique for each record and must be higher in the COBOL collating sequence than the value of the actual key used in the preceding WRITE statement. If the value of the actual key is out of sequence, the imperative statement following INVALID KEY will be executed.

Example:

Figure 6-2 shows how an ACTUAL KEY is used to sequentially update a record to an indexed file. Records are read in ascending order of their keys regardless of their positions in the file.

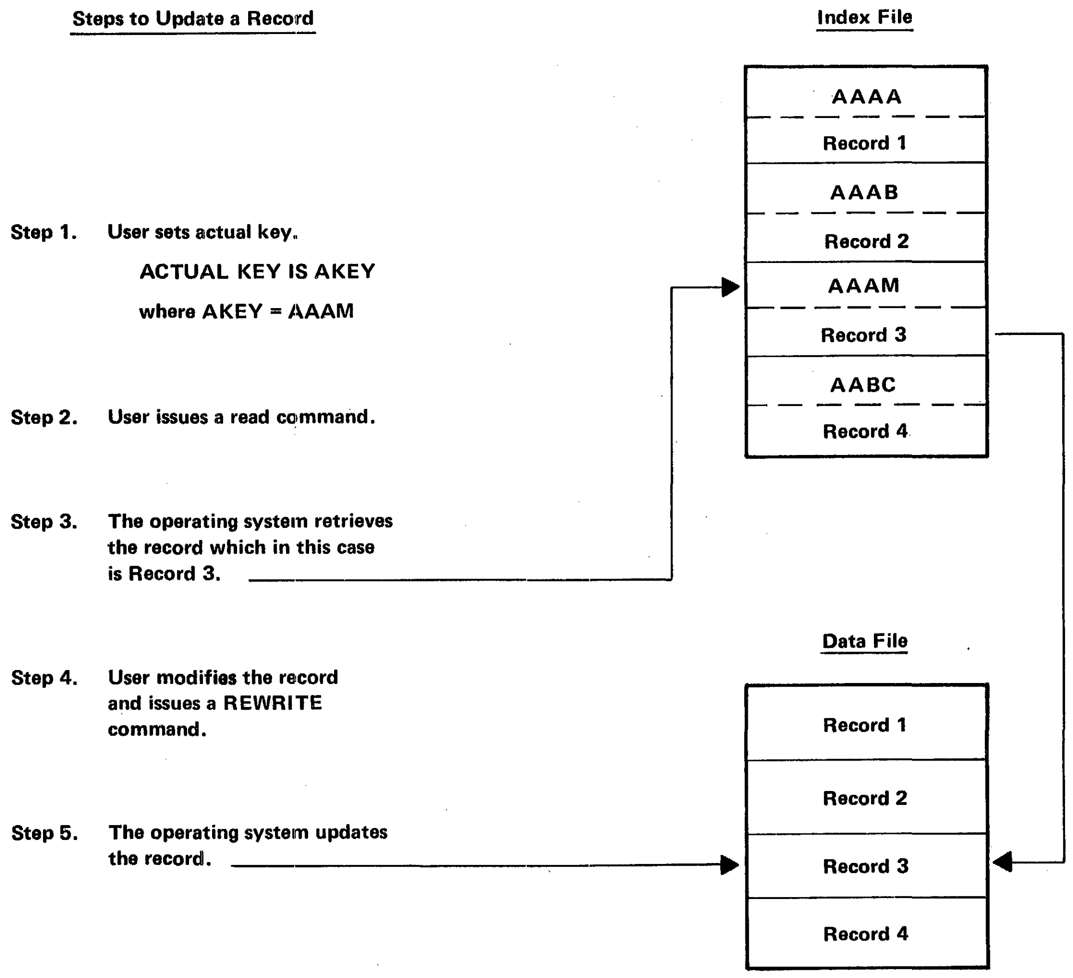

Steps to Update a Record
Index File
AAAA
Record 1
AAAB
Record 2
AAAM
Record 3
AABC
Record 4
Step 1. User sets actual key.
ACTUAL KEY IS AKEY
where AKEY = AAAM
Step 2. User issues a read command.
Step 3. The operating system retrieves the record which in this case is Record 3.
Step 4. User modifies the record and issues a REWRITE command.
Step 5. The operating system updates the record.
Data File
Record 1
Record 2
Record 3
Record 4

Figure 6-2. Sequential Access of an Indexed File

2. *Indexed file processed in random access mode.* Specification of an ACTUAL KEY clause is required when an indexed file is accessed randomly.

   The value of the actual key is used to locate a record with a matching key value when a record is retrieved (READ) or deleted (DELETE). If no match is found, the imperative statement following INVALID KEY is executed.

   The value of the actual key is used to create the index value or key value to be associated with the record when a record is added (WRITE). If the value of actual key already exists as an index-value in the file, the imperative statement following INVALID KEY is executed.

   The value of the actual key is used to check the validity of a record update (REWRITE). If the value of the actual key does not correspond to the value used for the preceding READ, an INVALID KEY condition exists.

Example:

Figure 6-3 shows how the actual key is used to locate a record during a read operation.

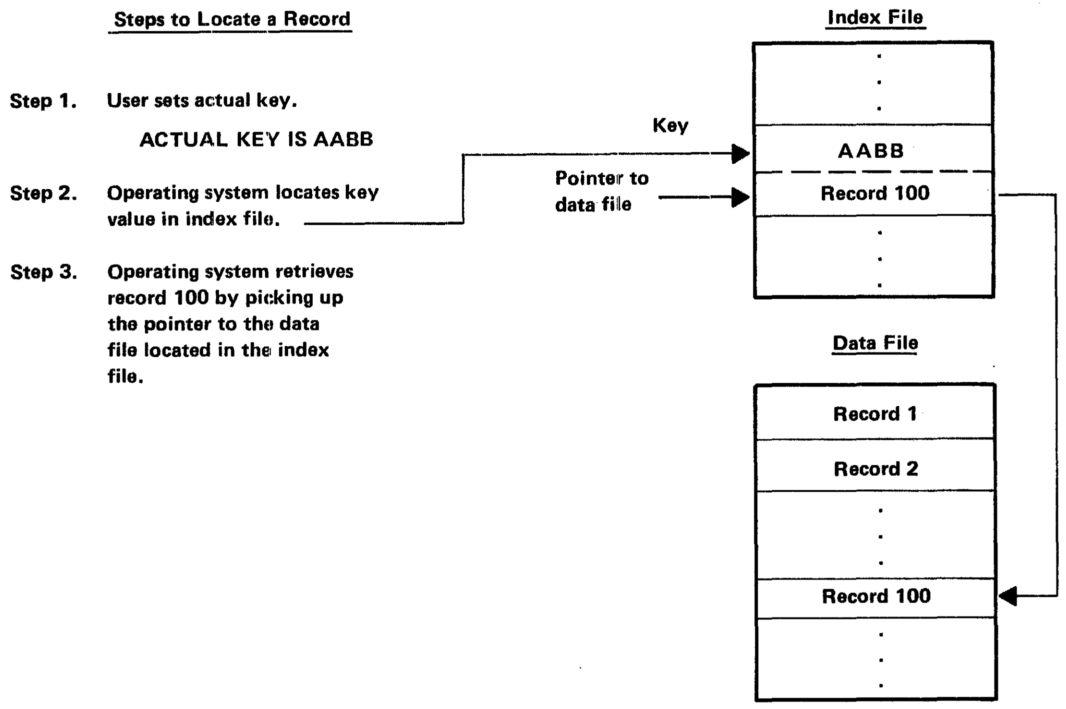

Steps to Locate a Record
Step 1. User sets actual key.
ACTUAL KEY IS AABB
Step 2. Operating system locates key value in index file.
Step 3. Operating system retrieves record 100 by picking up the pointer to the data file located in the index file.
Key
Pointer to data file
Index File
AABB
Record 100
Data File
Record 1
Record 2
Record 100

Figure 6-3. Random Access of an Indexed File

FORWARD KEY Clause

A FORWARD KEY clause is used with an indexed file when it is accessed sequentially. The format is as follows:

FORWARD KEY IS data-name

Data-name must specify a data-item that corresponds to the data-item specified by the ACTUAL KEY clause.

Specification of the FORWARD KEY clause is optional. If specified, the primary key value of the next record is placed in FORWARD KEY after a READ; that is, after a READ, a FORWARD KEY clause will contain the primary key value of the record that will be obtained in the next READ. This allows the user to add records to an index file with sequential access.

INDEX-BLOCK Clause

For indexed files the INDEX-BLOCK clause specifies the size (in characters) of the blocks within the index file. These index file blocks contain the keys. The format is as follows:

INDEX-BLOCK SIZE IS integer-1 CHARACTERS

The size of the index blocks is used at file creation time as the actual block size. This size overrides the block size determined via the Control Language //DEF statement.

The size of the index block is used by the compiler to allocate an area where the index blocks will be processed. When the file has been created, this size must be at least as large as the size of the actual index block.

Integer-1 must be a positive integer in the range 0   integer -1   7294 (a block cannot exceed one track).

Use Appendix G to determine the index file information such as key blocking factor and minimum block size.

When the key size and key blocking factor is known, the following formula (from Table 3 of Appendix G) is used to determine integer-1.

$$\text{Minimum block size} = 10 + \left(\frac{10 \times \text{MKB} \times (\text{KS}+4)}{9}\right)$$

(integer-1)

Where: MKB = minimum keys/block
KS = key size

FILE-CONTROL Clause Restrictions

Some of the clauses of the FILE-CONTROL paragraph are restricted in their use, and cause program errors if used incorrectly. Figure 6-4 shows these clauses and restrictions in the form of a matrix. Note, for example, that the FILE-LIMITS clause is illegal when specified for a unit record device.

	Key Attributes							
	Hardware Device			Organization			Access Mode	
SELECT Clauses	Disc	Tape	UR	Seq	Rel	Ind	Seq	Ran
MULTIPLE REEL	I	O	I	–	–	–	–	–
MULTIPLE UNIT	O	I	I	–	–	–	–	–
RESERVE	O	O	O	O	O	I	O	I
FILE-LIMITS	O	I	I	I	O	I	O	O
ACCESS IS SEQUENTIAL*	O	O	O	O	O	O	–	–
ACCESS IS RANDOM	O	I	I	I	O	O	–	–
PROCESSING IS SEQUENTIAL**	O	O	O	O	O	O	O	O
ACTUAL KEY	O	I	I	I	O	O	O	R
FORWARD KEY	O	I	I	I	I	O	O	I
INDEX-BLOCK SIZE	O	I	I	I	I	R	O	O

Organization			
Sequential*	O	O	O
Relative	O	I	I
Indexed	O	I	I

KEY

I	=	Illegal
R	=	Required
O	=	Optional
–	=	Not applicable

*Default is sequential
**Treated as comments

Figure 6-4. FILE-CONTROL Clause, Restrictions

Maximum FILE-CONTROL Paragraph

If the user chooses to specify all options available, the resultant structure of the FILE-CONTROL paragraph would appear as follows:

```
FILE-CONTROL.
   SELECT file-name
   ASSIGN TO [integer] system-name-1 [system-name-2]...
                   [FOR MULTIPLE {REEL}]
                                 {UNIT}
     RESERVE {integer}   ALTERNATE {AREA }
             {NO     }             {AREAS}
     FILE-LIMIT IS
     FILE-LIMITS ARE  integer-1 THRU integer-2
    [ACCESS MODE IS {SEQUENTIAL}]
                    {RANDOM    }
   [PROCESSING MODE IS SEQUENTIAL]
   [ACTUAL KEY IS data-name]
   [FORWARD KEY IS data-name]
   [INDEX-BLOCK SIZE IS integer-1 CHARACTERS].
```

I-O CONTROL PARAGRAPH

The I-O-CONTROL paragraph begins with the reserved word I-O-CONTROL followed by a period and a space.

The I-O CONTROL paragraph specifies the points at which rerun is to be established and the memory area which is to be shared by different files.

The I-O-CONTROL paragraph is optional. If used, the clauses must be in the specified order as follows:

```
I-O-CONTROL.
   [ [RERUN Clause]...
   [SAME AREA Clause]....]
```

RERUN Clause

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

```
RERUN ON system-name
   EVERY integer RECORDS OF file-name
```

The RERUN clause specifies that checkpoint records are to be written on the unit specified by system-name for every integer records of file-name that are processed. The value of integer must not exceed 65,535.

The system-name entry in this clause is used to specify the external medium of the file where the checkpoint records will be written. The structure of the system-name entry is identical to the structure of the name entry that appears in the ASSIGN clause. The system-name entry cannot duplicate any name entry previously used in an ASSIGN clause.

The name specified in the system-name entry for the checkpoint file must be the reserved identifier SYSCHK. The file organization used with this clause must be sequential and the class must *not* be unit-record.

It is possible to include several RERUN clauses within a single program. When multiple RERUN clauses are used, all checkpoint records are written on the SYSCHK file.

The integer entry is used to specify the number of READ, WRITE, DELETE, and REWRITE statements that occur in a single file. When the count of the READ, WRITE, DELETE, and REWRITE statements for a particular file equals the number specified in the integer entry for this clause, a checkpoint record is written.

When the checkpoints are written on disc, only the checkpoint immediately preceding the checkpoint being taken is saved. For example, checkpoint 1 is written on disc and saved, checkpoint 2 is then written on disc and saved. When checkpoint 3 is then written, checkpoint 1 is deleted and only checkpoint 2 remains on the disc.

When the checkpoints are written on tape, all checkpoints are saved.

After a checkpoint has been written, a message is printed on the operator console specifying the jobname and checkpoint number just completed. Refer to the **MRX/OS Control Program and Data Management Basic Reference** manual for further details.

Restart Considerations

The following conditions regarding restart procedures should be considered.

- The positioning of card input, card output, and printer files can be handled by the Checkpoint/Restart program if the spooling capabilities of MRX/OS are utilized.
- Tape files can be restarted, and they will be positioned.
- Mass storage files where the file organization is defined as SEQUENTIAL can be restarted, and they will be positioned.
- Mass storage files where the file organization is defined as RELATIVE can be restarted, but they are not positioned.

Restrictions and Limitations

The following conditions regarding the restrictions and limitations of the checkpoint/restart feature should be considered.

- On a deferred restart, no positioning is performed on the //PAR statement.
- Mass storage files where the file organization is defined as RELATIVE and the access mode is defined as RANDOM can be restarted, but any update records that have been made to the file must be handled by the user. They will not be handled by the Checkpoint/Restart program.
- Mass storage files where the file organization is defined as INDEX that are being created or updated cannot be restarted.

SAME AREA Clause

The SAME AREA clause specifies that two or more files are to use the same memory area during processing. Its format is as follows:

```
SAME AREA FOR file-name-1  file-name-2. . .
```

The area being shared includes all storage areas (including alternate areas) assigned to the files specified; therefore, it is not valid to have more than one of the files open at the same time.

More than one SAME AREA clause may be included in a program; however, a file-name must not appear in more than one SAME AREA clause, or more than once in a given SAME AREA clause.

Maximum I-O-CONTROL Paragraph

If the user wishes to specify all options available, the resultant structure of the I-O-CONTROL paragraph would appear as follows:

```
I-O-CONTROL.
   [RERUN ON system-name
    EVERY integer RECORDS OF file-name]. . .
   [SAME AREA FOR file-name-1  file-name-2]. . .
```

MAXIMUM INPUT-OUTPUT SECTION

If the user specified all the options available in the FILE-CONTROL and I-O-CONTROL paragraphs, an Input-Output Section is created that appears as follows:

```
[INPUT-OUTPUT SECTION.
FILE-CONTROL.
   SELECT file-name
   ASSIGN TO [integer] system-name-1 [system-name-2]. . .
                 [FOR MULTIPLE {REEL}]
                               {UNIT}
   [RESERVE {integer}    ALTERNATE {AREA }]
            {NO     }              {AREAS}
   [FILE-LIMIT IS      integer-1 THRU integer-2]
   [FILE-LIMITS ARE                            ]
   [ACCESS MODE IS {SEQUENTIAL}]
                   {RANDOM    }
   [PROCESSING MODE IS SEQUENTIAL]
   [ACTUAL KEY IS data-name]
   [FORWARD KEY IS data-name]
   [INDEX-BLOCK SIZE IS integer-1 CHARACTERS] . ]

[I-O-CONTROL.
  [RERUN ON system-name
        EVERY integer RECORDS OF file-name] . . .
   [SAME AREA FOR file-name-1  file-name-2] . . . ] . ]
```

MAXIMUM ENVIRONMENT DIVISION

If the user wishes to specify all options available the resultant structure of the Environment Division would appear as follows:

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. computer-name
OBJECT-COMPUTER. computer-name
┌                        ⎧ WORDS      ⎫ ┐
│ MEMORY SIZE integer    ⎨ CHARACTERS ⎬ │
└                        ⎩ MODULES    ⎭ ┘
  [SEGMENT-LIMIT IS priority-number.]
[SPECIAL-NAMES. [implementor-name IS mnemonic name] . . .
  [CURRENCY SIGN IS literal]
  [DECIMAL-POINT IS COMMA] .]
INPUT-OUTPUT SECTION.
[FILE-CONTROL.
  SELECT file-name
  ASSIGN TO [integer] system-name-1 [system-name-2] . . .
┌                  ⎧ REEL ⎫ ┐
└  FOR MULTIPLE    ⎩ UNIT ⎭ ┘
┌          ⎧ integer ⎫               ⎧ AREA  ⎫ ┐
└ RESERVE  ⎩ NO      ⎭  ALTERNATE    ⎩ AREAS ⎭ ┘
┌ FILE-LIMIT IS                                  ┐
└ FILE-LIMITS ARE    integer-1 THRU integer-2    ┘
┌                     ⎧ SEQUENTIAL ⎫ ┐
└ ACCESS MODE IS      ⎩ RANDOM     ⎭ ┘
  [PROCESSING MODE IS SEQUENTIAL]
  [ACTUAL KEY IS data-name]
  [FORWARD KEY IS data-name]
  [INDEX-BLOCK SIZE IS integer-1 CHARACTERS] .]
[I-O-CONTROL.
  [RERUN ON system-name
    EVERY integer RECORDS OF file-name] . . .
  [SAME AREA FOR file-name-1 file-name-2] . . .] .
```

7. DATA DIVISION

The Data Division describes the data that the object program will accept as input, manipulate or create. Data falls into two categories:

- Data contained in files. This type of data enters or leaves the internal memory of the computer from a specified area or areas.

- Data developed internally. This type of data is placed into intermediate or working storage.

The Data Division must be included in every COBOL source program as the third division.

ORGANIZATION OF THE DATA DIVISION

Three sections make up the Data Division: the File Section, the Working-Storage Section, and the Linkage Section. Each of these sections in the Data Division is optional.

The File Section defines the format of data files stored on an external device. Each file is defined by a file description (FD) followed by a record description or a series of record descriptions.

The Working-Storage Section describes records, data items and constants which are not part of external files. This data is developed and processed internally. The Working-Storage Section may specify both logical records and noncontiguous items.

The Linkage Section describes data that is made available from another program. Data item description entries and record description entries in the Linkage Section provide names and descriptions, but storage within the program is not reserved since the data area exists elsewhere. Any data description clause, except the VALUE clause, may be used to describe items in the Linkage Section.

The fixed names of these sections and the order of presentation are shown by the following format:

```
DATA DIVISION.
[FILE SECTION.
   {file description entry
   {record description entry } . . . } . . .]
[WORKING-STORAGE SECTION.
   [data item description entry] . . .
   [record description entry] . . .]
[LINKAGE SECTION.
   [data item description entry] . . .
   [record description entry] . . .]
```

DATA DIVISION ENTRIES

Each Data Division entry begins with a level indicator or a level number, followed by a space, the name of a data item, and a sequence of independent clauses describing the data item. The last clause is always terminated by a period followed by a space.

There are two types of Data Division entries: those which begin with a level indicator and those which begin with a level number.

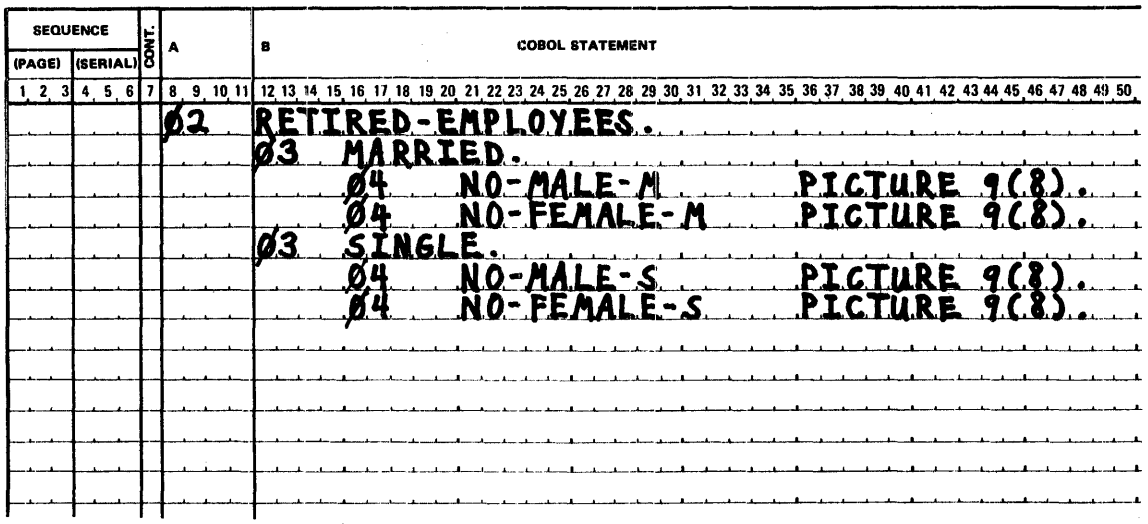
CONCEPT OF LEVELS

A level concept is inherent in the structure of a logical record. This concept arises from the need to specify subdivision 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.

The most basic subdivision of a record (that which is not further subdivided) is called an elementary item, 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 are combined into groups. Each group consists of a named sequence of one or more elementary items. Groups, in turn, may be combined into two or more groups. Thus, an elementary item may belong to more than one group.

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


SEQUENCE
(PAGE)
(SERIAL)
CONT.
A
B
COBOL STATEMENT

```
       Ø2  RETIRED-EMPLOYEES.
           Ø3  MARRIED.
               Ø4  NO-MALE-M         PICTURE 9(8).
               Ø4  NO-FEMALE-M       PICTURE 9(8).
           Ø3  SINGLE.
               Ø4  NO-MALE-S         PICTURE 9(8).
               Ø4  NO-FEMALE-S       PICTURE 9(8).
```

LEVEL NUMBERS

A system of level numbers shows the organization of elementary and group items.

Since records are the most inclusive data items, levels for record numbers start at 1 or 01. Less inclusive data items are assigned higher (not necessarily successive) level numbers not greater in value than 49. Separate entries are written in the source program for each level number used.

A group includes all group and elementary items following it until a level number numerically less than or equal to the level number of that group is encountered. The level number of an item (either an elementary item or a group item) immediately following the last elementary item of a group must be numerically equal to a previously stated group level number.

No true concept of levels exists for entries that specify noncontiguous Working-Storage items, which are not subdivisions of other items, and are not themselves subdivided, have been assigned the special level number 77. Level numbers 01 and 77 must begin in Area A, followed in Area B by associated data names and appropriate descriptive information.

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

LEVEL INDICATOR

The file description level indicator (FD) is used to specify the beginning of a file description entry in the File Section.

FILE SECTION

The File Section contains a description of all externally stored data (FD) used in the program.

The File Section must begin with the header FILE SECTION followed by a period. The File Section contains file description entries, each followed by its associated record description entry (or entries). The format is as follows.

```
FILE SECTION.
{file description entry
{record description entry} . . .}
```

FILE DESCRIPTION ENTRY

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

RECORD DESCRIPTION ENTRY

The Record Description entry consists of a set of data description entries which describe the particular record(s) contained within a particular file.

WORKING-STORAGE SECTION

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

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

```
WORKING-STORAGE SECTION.
[data item description entry] . . .
[record description entry] . . .
```

DATA ITEM DESCRIPTION ENTRIES

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

RECORD DESCRIPTION ENTRIES

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

LINKAGE SECTION

The Linkage Section describes data made available from another program (see *Subprogram Linkage Statements* in *Chapter 8. Procedure Division*). The format of this section is as follows.

```
LINKAGE SECTION.
[data item description entry] . . .
[record description entry] . . .
```

Data item description entries and record description entries in the Linkage Section provide names and descriptions, but storage within the program is not reserved since the data area exists elsewhere. Any data description clause except the VALUE clause may be used to describe items in the Linkage Section.

FILE DESCRIPTION ENTRY – DETAILS OF CLAUSES

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

FILE DESCRIPTION

The file description furnishes information concerning the physical structure, identification, and record names pertaining to a given file. Its general format is as follows:

```
FD file-name
   [BLOCK CONTAINS Clause]
   [RECORD CONTAINS Clause]
   [DATA RECORDS Clause]
   LABEL RECORDS Clause
   [VALUE OF Clause].
```

The level indicator, FD, identifies the beginning of a file description and must precede the file-name.

The clauses which follow the name of the file are optional in many cases, and their order of appearance is immaterial.

BLOCK CONTAINS CLAUSE

The BLOCK CONTAINS clause specifies the size of a physical record. Its format is as follows:

```
BLOCK CONTAINS integer {CHARACTERS}
                       {RECORDS   }
```

This clause is required excent when a physical record (BLOCK) contains one and only one complete logical record. If the clause is omitted it is assumed that records are blocked one record per block.

When the RECORDS option is used, the compiler assumes that the block size provides for integer records of maximum size plus additional space for any required control bytes. Integer must be a positive integer not greater than 255.

When the CHARACTERS option is used, the physical record size is specified in standard data format, that is, in terms of the number of bytes occupied internally. This is not necessarily the same as the number of characters used to represent the item within the physical record. The number of bytes occupied internally by a data item is included as part of the discussion of the USAGE clause in this section.

When the CHARACTERS option is used, integer represents the exact size of the physical record and must include slack bytes contained in the physical record. Each logical record contains a 4-byte control header which is transparent to the user, but must be taken into account when the CHARACTERS option is used. The integer specified must include the 4-byte control header. Integer must be a positive integer not larger than 2^{14}-1.

MRX COBOL does not allow logical records that extend across physical records.

RECORD CONTAINS CLAUSE

The RECORD CONTAINS clause specifies the size of data records. Its format is as follows:

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

Integer-1 and integer-2 must be positive integers not larger than 2^{14}-1. Integer-2 must be greater than integer-1.

Since the size of each data record is completely defined within the record description entry (which follows), this clause is never required. When present, however, the following rules apply:

- Integer-2 may not be used by itself unless all of the data records in the file have the same size. In this case, integer-2 represents the exact number of characters in the data record. If integer-1 and integer-2 are both shown, they refer to the minimum number of characters in the smallest size data record and the maximum number of characters in the largest size data record, respectively.
- The size is specified in terms of the number of characters in standard data format (refer to BLOCK CONTAINS clause). The size of a record is determined by the sum of the number of characters in all fixed length elementary items plus the sum of the maximum number of characters in all variable length items subordinate to the record.
- The 4-byte record header should not be included as part of the record size.

Whether this clause is specified or omitted, the record lengths are determined from the record descriptions. When a data item description entry within a record containing an OCCURS clause with the DEPENDING ON option, the compiler uses the maximum value of the variable to calculate the record length.

DATA RECORDS CLAUSE

The DATA RECORDS clause identifies the data records in a file by name. This clause serves only as documentation. Its format is as follows:

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

Both data-name-1 and data-name-2 are the names of data records and must have 01 level numbers.

The presence of more than one data-name indicates that the file contains more than one data record format. The multiple record descriptions for a given file will occupy the same storage area. The order in which they are listed is not significant.

LABEL RECORDS CLAUSE

The LABEL RECORDS clause specifies whether labels are present. Its format is as follows:

$$\underline{\text{LABEL}} \left\{ \begin{matrix} \underline{\text{RECORD}}\text{ IS} \\ \underline{\text{RECORDS}}\text{ ARE} \end{matrix} \right\} \left\{ \begin{matrix} \underline{\text{OMITTED}} \\ \underline{\text{STANDARD}} \end{matrix} \right\}$$

The LABEL RECORDS clause is required in every FD.

The OMITTED option specifies that either explicit labels do not exist for the file or the existing labels are nonstandard and the user wants to process them as data records. The OMITTED option should be specified for files assigned to unit record devices. It may be specified for files assigned to magnetic tape units.

The STANDARD option specifies that labels exist for the file, and that these labels conform to system specification. The STANDARD option should be specified for files assigned to disc units. It may be specified for files assigned to magnetic tape units.

VALUE OF CLAUSE

The VALUE OF clause particularizes the description of an item in the label records associated with a file. Its format is as follows:

$$\underline{\text{VALUE OF}}\text{ implementor-name-1 IS} \left\{ \begin{matrix} \text{data-name-1} \\ \text{literal-1} \end{matrix} \right\}$$
$$\left[\text{implementor-name-2 IS} \left\{ \begin{matrix} \text{data-name-2} \\ \text{literal-2} \end{matrix} \right\} \right] \ldots$$

A figurative constant may be substituted for any literal in the format. Implementor-names are standard label-field names. The following implementor-names are used to define field values in standard labels:

ID – a 17-byte alphanumeric, left justified, identification code

RETENTION-PERIOD – a 4-digit numeric, right-justified code indicating the length of time a file is to be kept

MODIFICATION-CODE – a 4-character alphanumeric, right-justified code used for access information.

This serves as documentation only since labels are specified by the //DEFINE statement in Control Language.

MAXIMUM FILE DESCRIPTION ENTRY

A file description entry including all options available would appear as follows:

```
FD  file-name
   [BLOCK CONTAINS integer {CHARACTERS}]
                           {RECORDS   }
   [RECORD CONTAINS [integer-1 TO] integer-2 CHARACTERS]
   [ DATA { RECORD IS   } data-name-1 [data-name-2]. . .]
          { RECORDS ARE }
   LABEL {RECORD IS  } {OMITTED }
         {RECORDS ARE} {STANDARD}
   [ VALUE OF implementor-name-1 IS {data-name-1}
                                    {literal-1  }
          [implementor-name-2 IS { data-name-2 }]. . . ]
                                 { literal-2   }
```

Clauses within the file description entry may appear in any order.

DATA DESCRIPTION

In COBOL, the terms used in connection with data description are as follows:

- Data description entry
- Data item description entry
- Record description entry

DATA DESCRIPTION ENTRIES

A data description entry specifies the characteristics of a particular item of data. The general format is:

```
level-number {data-name}
             {FILLER   }
[REDEFINES Clause]
[BLANK WHEN ZERO Clause]
[JUSTIFIED Clause]
[OCCURS Clause]
[PICTURE Clause]
[SYNCHRONIZED Clause]
[USAGE Clause]
[VALUE Clause].
```

The maximum length for a data description entry is 16,383 bytes. A data description entry is used for record description entries in the File, Working-Storage, and Linkage Sections and for data item description entries, in the Working-Storage and Linkage Sections. When used, the following rules apply:

- Level-number may be any number from 1-49 or 77.
- The clauses may be written in any order with two exceptions: the data-name or FILLER clause must immediately follow the level-number; the REDEFINES clause, when used, must immediately follow the data-name clause.
- The PICTURE clause must be specified for every elementary item, with the exception of index data items.
- Each entry must be terminated by a period.
- Successive data description entries may have the same format as the first or may be indented according to level number. 01 and 77 must start in Area A followed in Area B by record name or item name and appropriate descriptive information.

DATA ITEM DESCRIPTION ENTRIES

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

RECORD DESCRIPTION ENTRIES

A record description entry 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 if required, followed by a series of independent clauses as required. A record description entry has a hierarchical structure and, therefore, the clauses used with an entry may vary considerably, depending upon whether or not it is followed by subordinate entries. The structures of a record description is defined in *Concepts of Levels* in the beginning of this section. The elements allowed in a record description are shown in the data description entry general format.

DATA DESCRIPTION ENTRY CLAUSES

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

LEVEL NUMBER

The level number shows the hierarchy of data within a logical record. In addition it is used to identify entries for noncontiguous working-storage items. A level number is required as the first element in each data description entry.

Data description entries subordinate to an FD entry may have level numbers with the values 01-49. A single digit level number is written either as a space followed by a digit or as a zero followed by a digit.

The level number 01 identifies the first entry in each record description. Multiple level 01 entries subordinate to a FD level indicator in the File Section represent implicit redefinitions of the same area.

A special level number has been assigned to certain entries where there is no real concept of level: level number 77 is assigned to identify noncontiguous data items in Working-Storage and Linkage items.

DATA-NAME OR FILLER CLAUSE

A data-name specifies the name of the data being described. The word FILLER specifies an elementary or group item of the logical record that cannot be referred to directly. The general format of the clause is as follows:

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

In the File, Working-Storage, or Linkage Sections a data-name or the key word FILLER must be the first word following the level number in each data description entry. The key word FILLER may be used to name an elementary or group item in a record. Under no circumstances can a FILLER item be referred to directly.

Data-names must be unique; a name cannot be both a data-name and a procedure-name.

REDEFINES CLAUSE

The REDEFINES clause allows the same computer storage area to contain different data items. The format is as follows:

level-number data-name-1 <u>REDEFINES</u> data-name-2

When used, the REDEFINES clause must immediately follow data-name-1. The level numbers of data-name-1 and data-name-2 must be identical.

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

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

SEQUENCE (PAGE) (SERIAL) CONT. A B COBOL STATEMENT

```
       Ø2  A.
           Ø3  A-1  PICTURE   X.
           Ø3  A-2  PICTURE   XXX.
           Ø3  A-3  PICTURE   99.
       Ø2  B REDEFINES A PICTURE X(6).
```

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

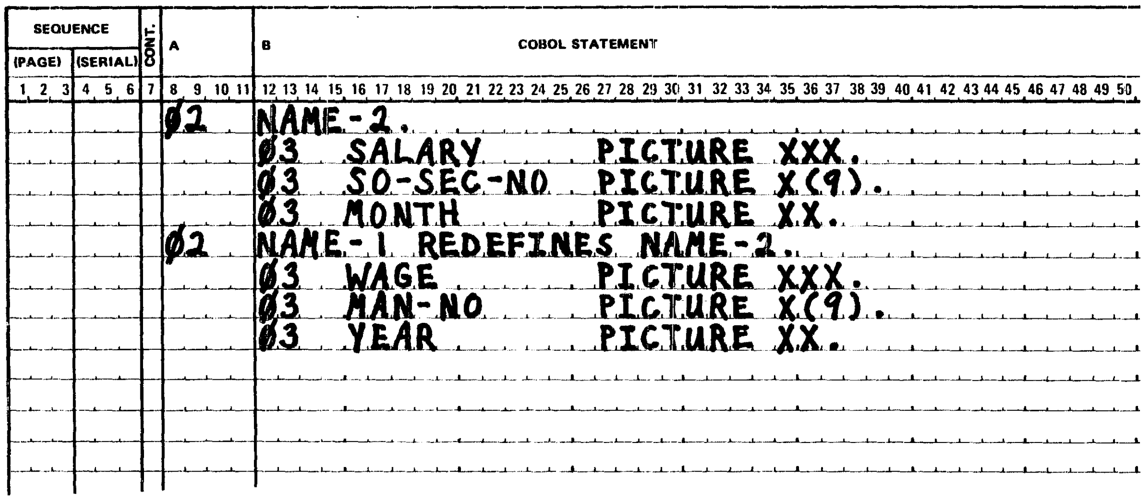
Multiple redefinitions of the same storage area are permitted, but each of these definitions must use the data-name of the entry that originally defined the area. Between redefined entries there can be no intervening entries that define new storage areas.

For example, in the following program segment, B, C, and D redefine A.

SEQUENCE (PAGE) (SERIAL) CONT. A B COBOL STATEMENT

```
       Ø2  A.
           Ø3  A-1          PICTURE XX.
           Ø3  A-2          PICTURE 99.
       Ø2  B REDEFINES A PICTURE X(4).
       Ø2  C REDEFINES A.
           Ø3  C-1  PICTURE X.
           Ø3  C-2  PICTURE 999.
       Ø2  D REDEFINES A PICTURE X(4).
```

Data items within an area can be redefined without their lengths being changed, as shown by the statements and resulting storage layout in Figure 7-1.


SEQUENCE
(PAGE)
(SERIAL)
CONT.
A
B
COBOL STATEMENT

```
Ø2  NAME-2.
    Ø3  SALARY       PICTURE XXX.
    Ø3  SO-SEC-NO    PICTURE X(9).
    Ø3  MONTH        PICTURE XX.
Ø2  NAME-1 REDEFINES NAME-2.
    Ø3  WAGE         PICTURE XXX.
    Ø3  MAN-NO       PICTURE X(9).
    Ø3  YEAR         PICTURE XX.
```

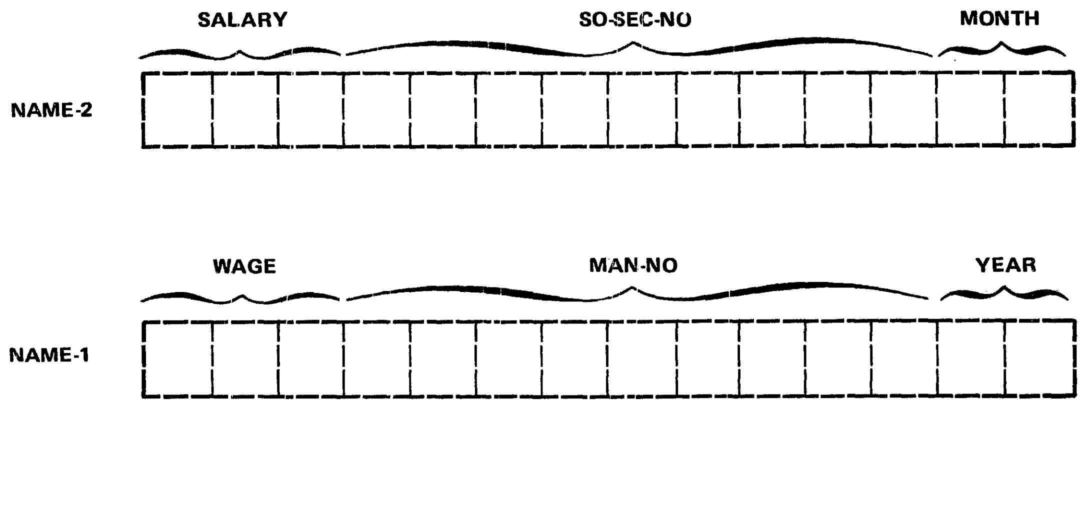


Figure 7-1. Data Items Redefined Within an Area

Data items can also be rearranged within an area, as shown by the statements and resulting storage layout in Figure 7-2.

```
       Ø2  NAME-2.
           Ø3  SALARY         PICTURE XXX.
           Ø3  SO-SEC-NO      PICTURE X(9).
           Ø3  MONTH          PICTURE XX.
       Ø2  NAME-1 REDEFINES NAME-2.
           Ø3  MAN-NO         PICTURE X(6).
           Ø3  WAGE           PICTURE 999V999
           Ø3  YEAR           PICTURE XX.
```

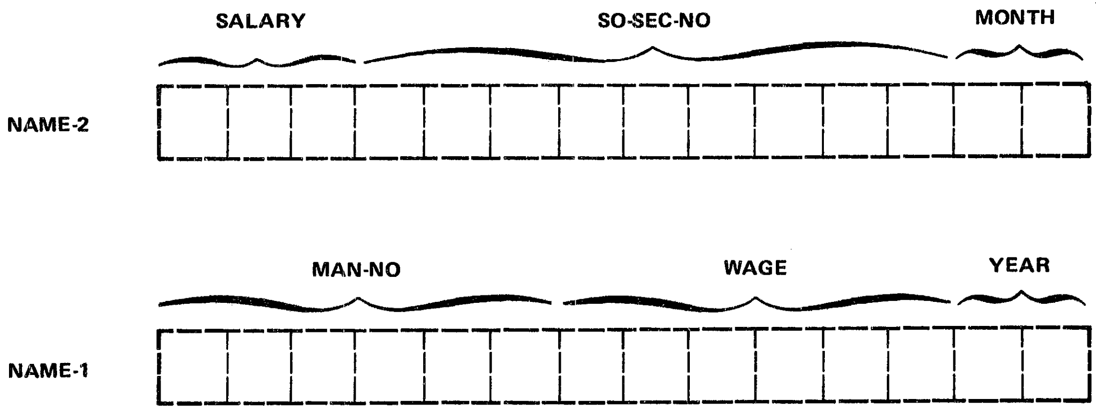
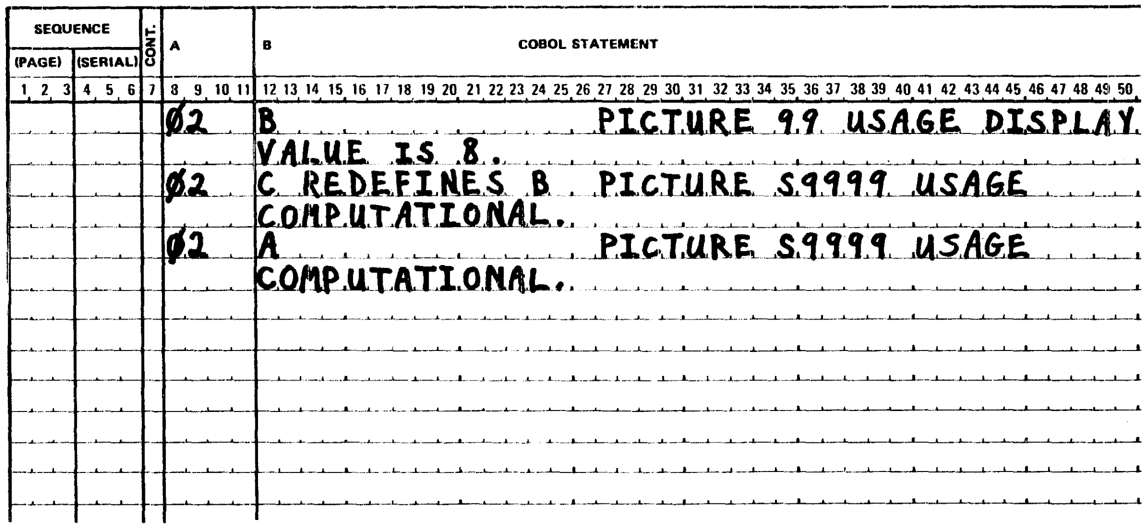


Figure 7-2. Data Items Rearranged Within an Area

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

The usage of data items within an area can be redefined. Altering the usage of an area through redefinition does not cause any change in existing data. Consider the example:



```
       Ø2  B                  PICTURE 99 USAGE DISPLAY
           VALUE IS 8.
       Ø2  C REDEFINES B  PICTURE S9999 USAGE
           COMPUTATIONAL.
       Ø2  A                  PICTURE S9999 USAGE
           COMPUTATIONAL.
```

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

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

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

```
Ø2  REGULAR-EMPLOYEE.
 Ø3 LOCATION   PIC A(8).
 Ø3 STATUS   PIC X(4).
 Ø3 SEMI-MONTHLY-PAY   PIC 9999V99.
 Ø3 WEEKLY-PAY REDEFINES SEMI-MONTHLY-PAY
    PIC 999V999.
Ø2  TEMPORARY-EMPLOYEE REDEFINES
    REGULAR-EMPLOYEE.
 Ø3 LOCATION-X   PIC A(8).
 Ø3 FILLER   PIC X(6).
 Ø3 HOURLY-PAY   PIC 99V99.
```

Following is a list of restrictions on the use of REDEFINES:

- The entries giving the new description of the storage area must not contain any VALUE clause.

- The data description entry for data-name-2 cannot contain a REDEFINES or an OCCURS clause, nor can data-name-2 be subordinate to an entry which contains a REDEFINES or an OCCURS clause.

- An item subordinate to data-name-2 may contain an OCCURS clause without the DEPENDING ON option. Data-name-1 or any items subordinate to data-name-1 may contain an OCCURS clause without the DEPENDING ON option. Neither data-name-2 nor data-name-1 nor any of their subordinate items may contain an OCCURS clause with the DEPENDING ON option. When data-name-1 has a level number other than 01, it must specify a storage area of the same size as data-name-2. For data-name-1 containing an OCCURS clause, the size of the storage area is computed by multiplying the length of one occurrence by the number of occurrences.

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

  ```
  02   A                  PICTURE X(4).
  02   B REDEFINES A PICTURE S9(8) COMP SYNC.
  ```

  he must ensure that A begins on a full word boundary.

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

BLANK WHEN ZERO CLAUSE

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

<u>BLANK</u> WHEN <u>ZERO</u>

When the BLANK WHEN ZERO clause is used, the item will contain nothing but spaces if the value of the item is zero, except if CHECK PROTECT is specified, BLANK WHEN ZERO is ignored.

The BLANK WHEN ZERO clause can be used only for an elementary item whose PICTURE is specified as numeric or numeric edited and USAGE is DISPLAY. When the BLANK WHEN ZERO clause is used for an item whose PICTURE is numeric, the category of the item is considered to be numeric edited.

Use of the BLANK WHEN ZERO clause is illegal when specified for data items having any of the following characteristics:

- PICTURE is alphanumeric, alphabetic, or alphanumeric edited
- USAGE is COMPUTATIONAL, COMPUTATIONAL-3, or INDEX
- Subordinate to a value description entry containing a VALUE clause
- Group item
- Specified in an OCCURS clause with the DEPENDING ON option

JUSTIFIED CLAUSE

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

$$\left\{\begin{matrix}\underline{\text{JUSTIFIED}}\\ \underline{\text{JUST}}\end{matrix}\right\}\ \text{RIGHT}$$

The standard rule for positioning data within an alphabetic or alphanumeric elementary item is as follows:

- The sending data is moved to the receiving character positions and aligned at the leftmost character position in the data item with space fill or truncation to the right.

When the receiving data item is described with the JUSTIFIED clause the positioning of data is as follows:

- When the sending data item is larger than the receiving data item, the leftmost characters are truncated.

- When the receiving data item is larger than the sending data item, the data is aligned at the rightmost character position in the data item with space fill to the left.

The JUSTIFIED clause can be specified only at the elementary item level. Use of the JUSTIFIED clause is illegal when specified for data items having any of the following characteristics:

- PICTURE is numeric, alphanumeric edited, or numeric edited

- USAGE IS COMPUTATIONAL, COMPUTATIONAL-3, or INDEX

- Group item

- Subordinate to a data description entry containing a VALUE clause

OCCURS CLAUSE

The OCCURS clause eliminates the need for separate entries for repeated data and supplies information required for the application of subscripts or indexes. The OCCURS clause is discussed in Section 9.

PICTURE CLAUSE

The PICTURE clause describes the general characteristics and editing requirements of an elementary item. The format is as follows:

$\left\{\begin{array}{l}\underline{\text{PICTURE}}\\ \underline{\text{PIC}}\end{array}\right\}$ IS character-string

A PICTURE clause can be used only 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.

The PICTURE clause must be specified for every elementary item except an index data item, for which this clause is prohibited from being used.

SYMBOLS USED IN THE PICTURE CLAUSE

The allowable symbols, used to describe an elementary item, and their functions are:

- A – Each 'A' in the character-string represents a character position which can contain only a letter of the alphabet or a space.

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

- 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.

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

- P – The 'P' indicates an assumed decimal scaling position and is used to specify the location of an assumed decimal point when the point is not within the number that appears in the data item. The scaling position character 'P' is not counted in the size of the data item. However, scaling position characters are counted in determining the maximum number of digit positions (18) in numeric edited items, or items which appear as operands in arithmetic statements. The scaling position character 'P' can appear only to the left or right of other characters 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's if 'P's are leftmost PICTURE characters or to the right of 'P's 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 sign character S and the assumed decimal point V are the only characters which may appear to the left of a leftmost string of P's.

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

- Z – Each 'Z' in a character-string is used to replace leftmost leading numeric character positions, that contain zero, with space characters. Each 'Z' is counted in the size of the item.

- 9 – Each '9' in the character-string represents a character position which can contain only a numeric character (0-9).

- 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 character ',' will be inserted. This character position is counted in the size of the item. The insertion character ',' must not be the last character in the PICTURE character-string.

- . – When the character '.' (period) appears in the character-string it is an editing symbol which represents the decimal point for alignment purposes and in addition, represents a character position into which the character '.' will be inserted. The 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 wherever they appear, in a PICTURE clause. The insertion character '.' must not be the last character in the PICTURE character-string.

- +, -, CR, DB – These symbols are used as editing sign control symbols. When used, they represent the character position into which the editing sign control symbol will be placed. The symbols are mutually exclusive in any one character-string and each character used in the symbol is counted in determining the size of the data-item.

- * – Each '*' (asterisk) 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 '$' (currency symbol) in the character-string represents a character position into which a currency symbol is to be placed. The currency symbol in a character-string is represented by either the symbol specified by '$' 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.

REPETITION OF SYMBOLS

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

A , X 9 P Z * B O + - $

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

```
A(40)
```

the (40) indicate forty consecutive appearances of the symbol A.

NOTE

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

```
S V . CR DB
```

CHARACTER STRING AND ITEM SIZE

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

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

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

In this compiler binary items occupy 2 or 4 bytes corresponding to PICTURE S9(4) and S9(8) respectively.

Therefore, in any operation such as the one described above, the compiler will truncate only such digits as would overflow a receiving field of 4 or 8 digits regardless of the PICTURE specified.

The result of the move in the foregoing example would be +2345 if the USAGE of the receiving field was COMPUTATIONAL (BINARY). (See Figure 7-3 for a detailed description of internal representation of binary items.)

FIVE CATEGORIES OF DATA

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

- Alphabetic
- Numeric
- Alphanumeric
- Alphanumeric edited
- Numeric edited

Alphabetic Items

An alphabetic item is one whose PICTURE character string contains only the symbol A. Its contents, when represented in standard data format, must be any combination of the 26 letters of the Roman alphabet and the space from the COBOL character set. Each alphabetic character is stored in a separate byte.

Alphanumeric Items

An alphanumeric item is one whose PICTURE character string is restricted to combinations of the symbols A, X, and 9. The item is treated as if the character string contained all X's. Its contents, when represented in standard data format, may be any of the allowable characters from the EBCDIC set.

A PICTURE character string which contains all A's or all 9's does not define an alphanumeric item.

Numeric Items

A numeric item is one whose PICTURE character string can only contain a valid combination of the following characters:

9, V, P, S

The contents of a numeric item, when represented in standard data format, must be a combination of the Arabic numerals 0-9 and must include an operational sign.

There are three types of numeric items: external decimal, binary, and internal decimal.

External Decimal

External Decimal corresponds to the form in which information is represented initially for card input or finally for printed or punched output. Decimal numbers in the zoned decimal format are external decimal items. Each digit of a number is represented by a single byte, with the four low-order bits of each eight-bit byte containing the value of the digit. The four high-order bits of each byte are zone bits; the zone bits of the least significant byte represent the sign of the item. Examples of external decimal items and their internal representation are shown in Figure 7-3. The maximum length of an external decimal item is 18 digits. USAGE IS DISPLAY is used in conjunction with external decimal items.

Item	Value	Description	Internal Representation*
External Decimal	-1234	DISPLAY PICTURE 9999	$\boxed{Z1}\boxed{Z2}\boxed{Z3}\underbrace{\boxed{F4}}_{\text{byte}}$
	+1234	DISPLAY PICTURE S9999	$\boxed{Z1}\boxed{Z2}\boxed{Z3}\underbrace{\boxed{D1}}_{\text{byte}}$ Note that, internally, the D4, which represents the -4, is the same bit configuration as the EBCDIC character M.
Binary	-1234	COMPUTATIONAL PICTURE S9999	$\underset{\overset{\uparrow}{S}}{\boxed{1111}}\boxed{1011}\underbrace{\boxed{0010}\boxed{1110}}_{\text{byte}}$ Note that, internally, negative binary numbers appear in two's complement form.
Internal Decimal	+1234	COMPUTATIONAL PICTURE 9999	$\boxed{01}\boxed{23}\underbrace{\boxed{4F}}_{\text{byte}}$
		COMPUTATIONAL PICTURE S9999	$\boxed{01}\boxed{23}\underbrace{\boxed{4C}}_{\text{byte}}$

*Codes used in this column are as follows.

Z = zone, equivalent to hexadecimal F, bit configuration 1111

Hexadecimal numbers and their equivalent meanings are:

F	=	nonprinting plus sign
C	=	internal equivalent of plus sign, bit configuration 1100
D	=	internal equivalent of minus sign, bit configuration 1101

S = sign position of a numeric field; internally,

1	=	in this position means the number is negative
0	=	in this position means the number is positive

Figure 7-3. Internal Representation of Numeric Items

Binary

A binary item has a decimal equivalent that consists of numeric characters 0 through 9, plus a sign. It occupies two bytes or four bytes, corresponding to specified decimal lengths in the PICTURE, of 4 digits and 8 digits, respectively. The leftmost bit of the storage area is the operational sign.

The PICTURE character string for a binary item may not contain the character P. The character V is legal only if it is the rightmost character of the PICTURE string. Binary items must be nonscaled integer numeric items and are always signed. Binary items with a PICTURE specifying 1 through 4 digits will be treated as if the PICTURE had specified 4 digits. Binary items with a PICTURE specifying 5 through 8 digits will be treated as if the PICTURE had specified 8 digits. A warning message will be given for binary item descriptions, if the PICTURE does not specify 4 or 8 digits. USAGE IS COMPUTATIONAL or USAGE IS BINARY must be specified for binary data items. An example of binary item and its internal representation is shown in Figure 7-3.

Internal Decimal

An internal decimal item consists of any of the numeric characters 0 through 9, plus a sign, and represents a value not exceeding 18 digits in length. It appears in storage in packed decimal format. That is, two digits per byte with the low-order four bits of the rightmost byte containing the sign. For items whose PICTURE does not contain an S, the sign position is occupied by a bit configuration which is interpreted as positive, but which does not represent an overpunch. Examples of internal decimal items and their internal representation are shown in Figure 7-3.

USAGE IS COMPUTATIONAL-3 or USAGE IS PACKED must be specified for internal decimal items.

Alphanumeric Edited Items

An alphanumeric edited item is one whose PICTURE character string is restricted to certain combinations of the following symbols:

A, X, 9, B, 0

To qualify as an alphanumeric edited item, one of the following conditions must be true:

- The character string must contain at least one B and at least one X.
- The character string must contain at least one 0 (zero) and at least one X.
- The character string must contain at least one 0 (zero) and at least one A. Its contents, when represented in standard data format, are allowable characters chosen from the EBCDIC set.

USAGE IS DISPLAY is used in conjunction with alphanumeric edited items. The maximum number of bytes allocated is 144.

Numeric Edited Items

A numeric edited item is one whose PICTURE character string is restricted to certain combinations of the symbols:

B P V Z 0 9 , . * + - CR DB $

The allowable combinations are determined from the order of precedence of symbols and editing rules. The maximum number of digit positions that may be represented in the character string is 18. The contents of the character positions that represent a digit, in standard data format, must be one of the numerals.

USAGE IS DISPLAY is used in conjunction with numeric edited items. The maximum number of bytes allocated is 144.

THREE CLASSES OF DATA

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

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

Figure 7-4 summarizes the relationships of the class and category for elementary and group data items.

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

Figure 7-4. Class and Category of Elementary and Group Data Items

EDITING RULES

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

- Simple insertion
- Special insertion
- Fixed insertion
- Floating insertion

There are two types of suppression and replacement editing:

- Zero suppression and replacement with spaces
- Zero suppression and replacement with asterisks

The type of editing which may be performed upon an item is dependent upon the category to which the item belongs. The following list 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

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.

Figure 7-5 shows examples of simple insertion editing.

Source PICTURE	Source Value	Edit PICTURE	Edited Result
9(5)	01234	99,999	01,234
X(9)	CITYSTATE	XXXXBXXXXX	CITY△STATE
X(10)	FMLASTNAME	XBXBXXXXXXXX	F△M△LASTNAME
X(5)	A3629	XBXXXX	A△3629
9(5)	00000	99,999	00,000

Figure 7-5. Examples of Simple Insertion Editing

Special Insertion Editing

The '.' (period) is used as the insertion character. In addition to being an insertion character it also represents 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 disallowed. 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.

Figure 7-6 shows examples of special insertion editing.

Source PICTURE	Source Value	Edit PICTURE	Edited Result
V9(6)	‸000135	.999999	.000135
9999V99	0013‸59	9999.99	0013.59
9999V99	0004‸28	99.9999	04.2800
9V9	2‸8	999.99	002.80
9999V9	1256‸4	99,999.99	01,256.30

Figure 7-6. Examples of Special Insertion Editing

Fixed Insertion Editing

The currency symbol and the editing sign control symbols, '+', '-', 'CR', '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 of the item. The symbol '+' or '-', when used, may occupy the leftmost or the rightmost character position of the item. The currency symbol must be the leftmost character position 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. Editing sign control symbols produce the following results, as shown in Figure 7-7, depending upon the value of the data item.

Figure 7-8 shows examples of fixed insertion editing.

Editing Symbol in PICTURE Character-String	Result	
	Positive or Zero	Data Item Negative
+	+	–
-	Space	–
CR	2 Spaces	CR
DB	2 Spaces	DB

Figure 7-7. Editing Sign Control Symbols and Results

Source PICTURE	Source Value	Edit PICTURE	Edited Result
99V99	01‸23	$99.99	$01.23
S99V99	-01‸23	+99.99	-01.23
S99V99	+01‸23	+99.99	+01.23
S99V99	-01‸23	-99.99	-01.23
S99V99	+01‸23	-99.99	Δ01.23
S9999	3921	9999-	3921Δ
999V99	123‸45	$999.99CR	$123.45ΔΔ
S999V99	-123‸45	$999.99CR	$123.45CR
999V99	123‸45	$999.99DB	$123.45ΔΔ
S999V99	-123‸45	$999.99DB	$123.45DB

Figure 7-8. Examples of Fixed Insertion Editing

Floating Insertion Editing

The currency symbol and editing sign 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.

In a 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 on 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 in the character position immediately preceding the decimal point, or the first nonzero digit in the data represented by the insertion symbol string, whichever is further 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 character.

Figure 7-9 shows examples of floating insertion editing.

Source PICTURE	Source Value	Edit PICTURE	Edited Result
99V99	12‸34	$$$$.99	$12.34
9999	1234	$$$$.99	$234.00
V9999	‸1234	$$$$.99	$.12
S99V99	+12‸34	----.99	12.34
S9V999	-1‸234	----.99	-1.23
9V999	1‸234	$$99.99	$01.23
9999	0000	----	ΔΔΔΔ

Figure 7-9. Examples of Floating Insertion Editing

Zero Suppression and Replacement Editing

The suppression of leading zeros in numeric character positions is indicated by the use of the alphabetic 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 character will be the space and if the asterisk is used, the replacement character will be '*'.

Zero suppression and replacement is indicated in a PICTURE character-string by using a string of one or more of the allowable symbols to represent leading numeric character positions which are to be replaced when the 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 any or all of the leading numeric character positions to the left of the decimal point by suppression symbols. 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 item which corresponds to a symbol in the string is replaced by the replacement character. Suppression terminates at the first nonzero digit in the data item 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'. If the suppression symbol is '*' and the data is zero, the entire data item will be '*' except for the actual decimal point which will be printed.

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.

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

Figure 7-10 shows examples of zero suppression and replacement editing.

Source PICTURE	Source Value	Edit PICTURE	Edited Result
99V99	10‸34	****.99	**10.34
99V99	10‸34	ZZZZ.99	ΔΔ10.34
99V99	00‸00	ZZZZ.99	ΔΔΔΔ.00
99V99	00‸00	ZZZZ.ZZ	ΔΔΔΔΔΔΔ
99V99	00‸00	****.**	****.**
9999V999	1034‸567	ZZZZ.ZZ	1034.56
99V99	10‸34	Z999.99	Δ010.34
9V99	0‸00	$*,***.99CR	$*****.00ΔΔ

Figure 7-10. Zero Suppression and Replacement Editing

PICTURE CLAUSE RESTRICTIONS

In general, usage of the PICTURE clause is restricted as follows:

- Illegal when specified for a data item having USAGE IS INDEX
- Illegal when specified for a group item
- Required for elementary item whose USAGE is not INDEX

Use of any of the following clauses is illegal when specified for an item having an alphanumeric PICTURE specification:

- BLANK WHEN ZERO
- USAGE IS COMPUTATIONAL, COMPUTATIONAL-3, or INDEX
- VALUE IS numeric literal

Use of any of the following clauses is illegal when specified for an elementary item having an alphabetic PICTURE specification:

- BLANK WHEN ZERO
- USAGE IS COMPUTATIONAL or COMPUTATIONAL-3
- VALUE IS numeric literal

Use of any of the following clauses is illegal when specified for an elementary item having a numeric PICTURE specification:

- JUSTIFIED
- VALUE IS nonnumeric literal

Use of any of the following clauses is illegal when specified for an elementary item having an alphanumeric edited PICTURE specification:

- BLANK WHEN ZERO
- JUSTIFIED
- USAGE IS COMPUTATIONAL or COMPUTATIONAL-3
- VALUE IS numeric literal

Use of any of the following clauses is illegal when specified for an elementary item having a numeric edited PICTURE specification:

- JUSTIFIED
- USAGE IS COMPUTATIONAL or COMPUTATIONAL-3
- VALUE IS numeric literal

SYNCHRONIZED CLAUSE

The SYNCHRONIZED clause specifies the alignment of an elementary item on a word boundary in core storage. The effect of this is to ensure efficiency in the performance of arithmetic operations on binary data. The format is as follows:

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

If either the LEFT or RIGHT option is specified it is treated as a comment.

The SYNCHRONIZED clause may appear only at the elementary level. The length of the item synchronized will not be affected by the use of this clause.

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

When the item is aligned, the character position between the last item assigned and the current item is known as "slack byte". This unused character position is included in the size of any group to which the elementary item preceding the synchronized elementary item belongs.

When a COMPUTATIONAL item is SYNCHRONIZED, it is aligned on a word boundary. When a DISPLAY or COMPUTATIONAL-3 item is SYNCHRONIZED, the SYNCHRONIZED clause is treated as a comment.

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

```
02     A                            PICTURE X(4).
02     B  REDEFINES A PICTURE S9(8) COMP SYNC.
```

he must ensure that A begins on a word boundary.

When SYNCHRONIZED is *not* specified for binary items, no space is reserved for slack bytes. However, when computation is done on these fields, the compiler generates the necessary instructions to move the items to a work area which has the correct boundary necessary for computation.

SLACK BYTES

There are two types of slack bytes: intra-record slack bytes and inter-record slack bytes. An intra-record slack byte is an unused character position preceding a synchronized item in the record, or an unused character position added between table entries containing synchronized items. An inter-record slack byte is an unused character position added between blocked logical records.

Intra-Record Slack Bytes

For an output file, or in the Working-Storage Section, the compiler inserts intrarecord slack bytes to ensure that all SYNCHRONIZED items are on their proper boundaries. For an input file, the compiler expects intra-record slack bytes to be present when necessary to assure the proper alignment of a SYNCHRONIZED item.

Because it is important for the user to know the length of the records in a file, the algorithm the compiler uses to determine whether a slack byte is required is as follows:

> The total number of bytes occupied by all elementary data items preceding the synchronized binary item are added together, including any slack bytes previously added. If this sum is even, no slack byte is added. If the sum is an odd number, one slack byte is added.

This intra-record slack byte is added to each record immediately following the elementary item preceding the synchronized binary item. It is defined as if it were an item with a level number equal to that of the elementary item that immediately precedes the SYNCHRONIZED item, and is included in the size of the group which contains it.

For example:

```
01    FIELD-A.
      02    FIELD-B           PICTURE X95).
      02    FIELD-C.
             03    FIELD-D    PICTURE XX.
            [03    Slack-Byte PIC X. Inserted by compiler]
             03    FIELD-E    PICTURE S9(8) COMP SYNC.

01    FIELD-L.
      02    FIELD-M           PICTURE X(5).
      02    FIELD-N           PICTURE XX.
     [02    Slack-Byte        PIC X. Inserted by compiler]
      02    FIELD-O.
             03    FIELD-P    PICTURE S9(8) COMP SYNC.
```

Slack bytes may also be added by the compiler when a group item is defined with an OCCURS clause and contains within it a synchronized data item with USAGE defined as COMPUTATIONAL.

To determine whether a slack byte is to be added, the compiler calculates the size of the group, including all the necessary intra-record slack bytes. If this sum is an even number of bytes, no slack byte is added. If the sum is an odd number of bytes, one slack byte will be added at the end of each occurrence of the group containing the OCCURS clause.

For example, a record is defined as follows:

```
01      WORD-RECORD
        02      WORK-CODE PICTURE X.
        02      COMP-TABLE OCCURS 10 TIMES.
                 03   COMP-TYPE PICTURE XX.
                [03   IR-Slack byte PIC X. Inserted by compiler]
                 03   COMP-HRS PICTURE S9(4) COMP SYNC.
                 03   COMP-NAME PICTURE X(6).
```

In order to align COMP-HRS upon its proper boundary, the compiler has added one slack byte. However, without further adjustments, the second occurrence of COMP-TABLE would now begin on a word boundary, making the insertion of a slack byte unnecessary. This would create a table entry with a size different from that of the first entry, and create addressing problems.

In order to make all table entries the same size, the compiler must add an inter-record slack byte at the end of the group, as though the record had been written:

```
01      WORK-RECORD.
        02      WORK-CODE PIC X.
        02      COMP-TABLE OCCURS 10 TIMES
                 03   COMP-TYPE PIC XX.
                [03   IR-Slack byte PIC X. Inserted by compiler]
                 03   COMP-HRS PIC S9(3) COMP SYNC.
                 03   COMP-NAME PIC X(6).
                [03   IR-Slack byte PIC X. Inserted by compiler]
```

Using this description, the second (and each succeeding) occurrence of COMP-TABLE begins on a byte boundary and has the same storage layout as the first. Figure 7-11 shows the storage layout for the first and the second occurrences of COMP-TABLE.

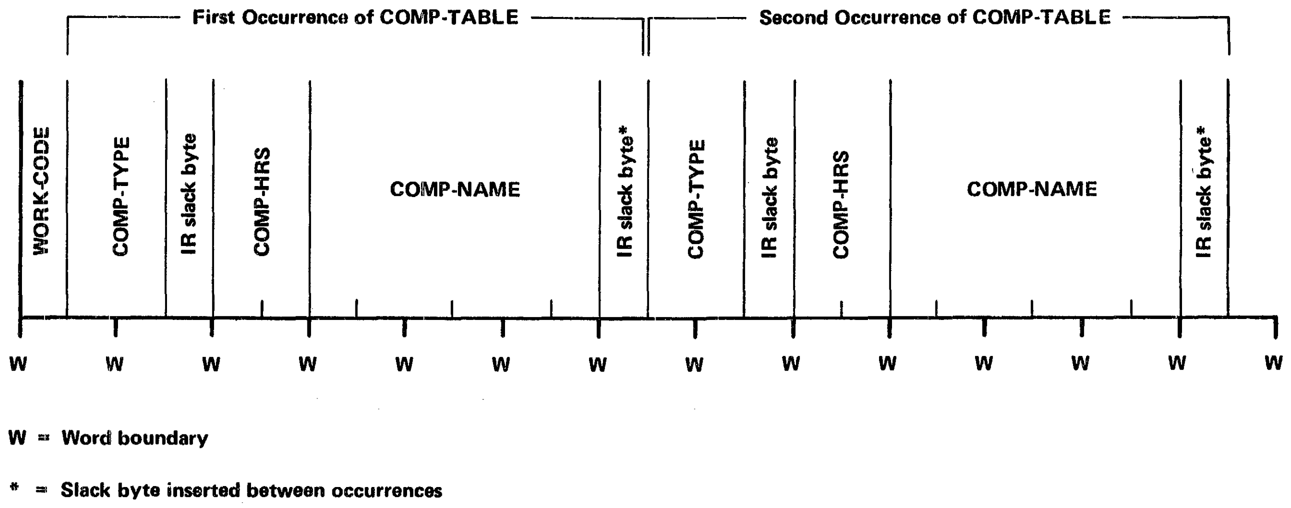


Figure 7-11. Insertion of Slack Bytes Between Occurrences

Each succeeding occurrence within the table will now begin at the same relative position to word boundaries as the first.

Inter-Record Slack Bytes

If the file contains blocked logical records, the logical records must contain an even number of bytes, including all intra-record slack bytes created. It is the user's responsibility to insure that the logical record length is equal to an even number of bytes. An inter-record slack byte may be specified by writing a FILLER at the end of the record.

SYNCHRONIZED CLAUSE RESTRICTIONS

Use of the SYNCHRONIZED clause is illegal when specified for a data item having any of the following characteristics:

- USAGE IS INDEX
- Group item
- Subordinate to a VALUE clause

USAGE CLAUSE

The USAGE clause specifies the format of a data item in the computer storage. The format is as follows:

$$[\underline{\text{USAGE}}\ \text{IS}]\left\{\begin{array}{l}\underline{\text{INDEX}}\\ \underline{\text{DISPLAY}}\\ \underline{\text{COMP}}\\ \underline{\text{COMPUTATIONAL}}\\ \underline{\text{BINARY}}\\ \underline{\text{COMP-3}}\\ \underline{\text{COMPUTATIONAL-3}}\\ \underline{\text{PACKED}}\end{array}\right\}$$

The USAGE clause can be written at any level. If the USAGE clause is written at a group level, it applies to each elementary item in the group. The USAGE clause of an elementary item cannot contradict the USAGE clause of a group to which the item belongs. If the USAGE clause is not specified for an elementary item, or for any group to which the item belongs, it is assumed that the USAGE is DISPLAY.

This clause specifies the manner in which a data item is represented in the storage of a computer. It does not affect the use of the data item, although the specifications for some statements in the Procedure Division may restrict the usage of the operands referred to. The USAGE clause may affect the radix or type of character representation of the item.

The DISPLAY option specifies that the item is stored in character form, one character per byte. DISPLAY is used in conjunction with alphabetic, alphanumeric, external decimal, alphanumeric edited, and numeric edited items. The allowable characters used to represent these items can be found in the discussion of the PICTURE Clause.

The COMPUTATIONAL option specifies an integer binary data item occupying two or four bytes, corresponding to specified digit lengths of 1-4, and 5-8, respectively. For example, an item whose PICTURE is S9(5) and whose USAGE is COMPUTATIONAL has an internal length of 4 bytes.

The PICTURE of a COMPUTATIONAL item may contain only 9's, the operational sign character S, and the character V only if the V is the rightmost character of the picture string.

NOTE

> Regardless of the PICTURE specified for a COMPUTATIONAL item, this compiler will treat the item as if the PICTURE were specified as S9(4) and S9(8), respectively.

The COMPUTATIONAL-3 option specifies that the item is stored in packed decimal format: two digits per byte, with the low-order four bits of the rightmost byte containing the sign.

The PICTURE of a COMPUTATIONAL-3 item may contain only 9's and the operational sign character S, the implied decimal value V, and one or more P's.

A COMPUTATIONAL or COMPUTATIONAL-3 item represents a value to be used in computations and must be numeric. If a group item is described as COMPUTATIONAL or COMPUTATIONAL-3, it is the elementary items in the group which have this USAGE. The group item itself cannot be used in computations (see discussion of numeric data items in the *PICTURE Clause* in this section).

USAGE IS INDEX is discussed in *Table Handling* at the beginning of Chapter 9.

Use of the USAGE IS COMPUTATIONAL or USAGE IS COMPUTATIONAL-3 clauses is illegal when specified for data items having any of the following characteristics:

- BLANK WHEN ZERO
- JUSTIFIED
- PICTURE is alphabetic, alphanumeric, alphanumeric edited, or numeric edited
- VALUE IS alphanumeric literal
- Subordinate to a VALUE clause

Use of the USAGE IS INDEX clause is illegal when specified for data items having any of the following characteristics:

- BLANK WHEN ZERO
- JUSTIFIED

- PICTURE
- SYNCHRONIZED
- VALUE
- Subordinate to a VALUE Clause

VALUE CLAUSE

The VALUE clause defines the initial value of working-storage items. The format is as follows:

```
VALUE IS literal
```

A figurative constant may be substituted wherever a literal is specified.

The VALUE clause must not be stated for any item whose size, explicit or implicit, is variable.

Rules governing the use of the VALUE clause differ with the particular section of the Data Division in which it is specified:

- In the File Section and the Linkage Section, the VALUE clause must not be used.
- In the Working-Storage Section, the VALUE clause may be used to specify the initial value of any data item. It causes the item to assume the specified value at the start of execution of the object program. If the VALUE clause is not used in an item's description, the initial VALUE is unpredictable.

The VALUE clause must not be specified in a data description entry that contains an OCCURS clause or in an entry that is subordinate to an entry containing an OCCURS clause.

Within a given record description, the VALUE clause must not be used in a data description entry following a data description entry which contains an OCCURS clause with a DEPENDING ON phrase.

The VALUE clause must not be specified in a data description entry which contains a REDEFINES clause or in an entry which is subordinate to an entry containing a REDEFINES clause.

If the VALUE clause is used in an entry at the group level, the literal must be a figurative constant or a nonnumeric literal, and the group area is initialized without consideration for the USAGE of the items contained within this group. The VALUE clause then cannot be specified at subordinate levels within this group.

The VALUE clause cannot be specified for a group containing items with descriptions including JUSTIFIED, SYNCHRONIZED, or USAGE (other than USAGE IS DISPLAY) clauses.

The VALUE clause must not conflict with the other clauses in the data description of the item or in the data description within the hierarchy of the item. If the category of an elementary item is specified as numeric or alphabetic, it does not contradict the alphanumeric category of group items.

The following rules apply:

- If the item is numeric, the literal in the VALUE clause must be a numeric literal. If the literal defines the value of a Working-Storage item, the literal is aligned according to the rules for numeric moves, except that the literal must not have a value that would require truncation of nonzero digits.

- If the item is alphabetic or alphanumeric, the literal in the VALUE clause must be a nonnumeric literal. The literal is aligned according to the alignment rules (see *JUSTIFIED Clause* in this section) except that the number of characters in the literal must not exceed the size of the item.

- The numeric literal in a VALUE clause of an item must have a value that is within the range of values indicated by the PICTURE clause for that item. For example, for PICTURE 99PPP, the literal must be within the range 1000 through 99000 or zero. For PICTURE PPP99, the literal must be within the range .00000 through .00099.

- If the item is numeric edited or alphanumeric edited, the literal in the VALUE clause must be a nonnumeric literal already in edited form. The editing characters in a PICTURE clause are ignored in determining the initial appearance of the item described, but they are included in determining the size of the item.

MAXIMUM RECORD OR DATA ITEM DESCRIPTION ENTRY

A record or data item description entry including all options available would appear as follows (the OCCURS clause is discussed in Chapter 9):

```
             {data-name-1}
level number {FILLER     }
    [REDEFINES data-name-2]
    [BLANK WHEN ZERO]
    [{JUSTIFIED}       ]
    [{JUST     }  RIGHT]
    [OCCURS Clause]
    [{PICTURE}                     ]
    [{PIC    }  IS character string]
    [{SYNCHRONIZED}   [LEFT ]]
    [{SYNC        }   [RIGHT]]
    [                {INDEX          } ]
    [                {DISPLAY        } ]
    [                {COMP           } ]
    [  USAGE IS      {COMPUTATIONAL  } ]
    [                {BINARY         } ]
    [                {COMP-3         } ]
    [                {COMPUTATIONAL-3} ]
    [                {PACKED         } ]
    [VALUE IS literal]
```

8. PROCEDURE DIVISION

The Procedure Division must be included in every COBOL source program. It contains the specific instructions for solving a data processing problem. These instructions are written in procedures.

ORGANIZATION OF THE PROCEDURE DIVISION

The Procedure Division contains procedures. A procedure is composed of a paragraph, or a group of successive paragraphs, or a section, or a group of successive sections within the Procedure Division. If one paragraph is in a section, then all paragraphs must be in sections. A procedure-name is a word used to refer to a paragraph or section in the source program in which it occurs. It consists of a paragraph-name or a section-name.

The end of the Procedure Division and the physical end of the program is that physical position in a COBOL source program after which no further procedures appear.

The Procedure Division must begin with the header PROCEDURE DIVISION followed by a period and a space unless subprogram linkage is used. In this case, the Procedure Division header in a called program may optionally contain a USING clause preceding the period (see *Subprogram Linkage* in this section).

A section consists of a section header followed by one or more successive paragraphs. A section ends immediately before the next section-name or at the end of the Procedure Division.

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.

The structure of the Procedure Division is as follows:

```
PROCEDURE DIVISION. [USING identifier-1[identifier-2] . . .].
{section-name SECTION priority .
{paragraph-name. {sentence} . . . } . . . } . . .
```

STATEMENTS

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

COMPILER DIRECTING STATEMENTS

A compiler directing statement directs the compiler to take a specific action. The statements consist of a compiler directing verb and its operands. The compiler directing verbs are ENTER and NOTE.

CONDITIONAL STATEMENTS

A conditional statement causes the program to select alternate paths of control depending upon the truth value of a test.

COBOL statements used as conditional statements are:

```
IF        \
ADD        |
SUBTRACT   >  (ON SIZE ERROR)
MULTIPLY   |
DIVIDE    /

GO            (DEPENDING ON)

READ          (AT END)

READ      \
WRITE      |
DELETE     >  (INVALID KEY)
START      |
REWRITE   /
```

The options in parentheses cause otherwise imperative statements to be treated as conditionals at execution time. A discussion of these options is included as part of the description of the associated imperative statement.

IMPERATIVE STATEMENTS

An imperative statement indicates a specific action to be taken by the object program. An imperative statement is any statement that is neither a conditional statement nor a compiler-directing statement. An imperative statement may consist of a sequence of imperative statements.

COBOL verbs used in imperative statements are grouped into the following categories and subcategories:

Arithmetic

ADD
SUBTRACT
MULTIPLY
DIVIDE

Procedure Branching

GO TO
ALTER
PERFORM
EXIT
STOP

Data Manipulation

MOVE
EXAMINE

Input/Output

ACCEPT
CLOSE
DISPLAY
OPEN
READ
WRITE
SEEK
DELETE
START
REWRITE

Table Handling (discussed in Section 9)

SET

Subprogram Linkage

CALL
EXIT (PROGRAM)

SENTENCES

A compiler directing sentence is a single compiler directing statement terminated by a period, followed by a space.

An imperative sentence is an imperative statement or a series of imperative statements terminated by a period, followed by a space.

A conditional sentence is a conditional statement optionally preceded by an imperative statement terminated by a period, followed by a space.

CONDITIONS

A condition is one of the following:

- Relation condition
- Class condition
- NOT condition

The construction (NOT condition) where condition is one of the conditions listed above, is not permitted if the condition itself contains a NOT.

TEST CONDITIONS

A test condition is an expression that, taken as a whole, may be either true or false, depending on the circumstances existing when the expression is evaluated.

There are two types of simple conditions which, when preceded by the word IF, constitute one of the two types of tests: class test, and relation test.

The construction — NOT condition — may be used in any simple test condition to make the relation specify the opposite of what it would express without the word NOT. For example, NOT AGE GREATER THAN 21 is the opposite of AGE GREATER THAN 21.

Each of the previously mentioned tests, when used within the IF statement, constitutes a conditional statement.

CLASS CONDITION

The class test determines whether data is alphabetic or numeric. The format for a class condition is as follows.

$$\text{identifier IS}[\underline{\text{NOT}}]\left\{\begin{array}{l}\underline{\text{NUMERIC}}\\ \underline{\text{ALPHABETIC}}\end{array}\right\}$$

The operand being tested must be implicitly or explicitly described as USAGE DISPLAY.

A numeric data item consists of the digits 0 through 9, with or without an operational sign.

An alphabetic data item consists of the space character and the characters A through Z.

The identifier being tested is determined to be alphabetic only if the contents consist of any combination of the alphabetic characters A through Z and the space.

If the PICTURE in the record description of the identifier being tested does not contain an operational sign, the identifier being tested is determined to be numeric only if the contents are numeric and an operational sign is not present.

The NUMERIC test cannot be used with an identifier described as alphabetic.

The ALPHABETIC test cannot be used with an identifier described as numeric.

Figure 8-1 shows allowable forms of the class test.

Identifier Type	Allowable Class Test	
Alphabetic	ALPHABETIC	NOT ALPHABETIC
Alphanumeric	ALPHABETIC NUMERIC	NOT ALPHABETIC NOT NUMERIC
Numeric	NUMERIC	NOT NUMERIC

Figure 8-1. Allowable Forms of the Class Test

RELATION CONDITION

A relation condition causes a comparison of two operands, each of which may be an identifier or a literal. Comparison of two numeric operands is permitted regardless of the format as specified in individual USAGE clauses. However, for all other comparisons, the operands must have the same usage.

The general format for a relation condition is as follows:

$$\left\{\begin{matrix}\text{identifier-1}\\ \text{literal-1}\end{matrix}\right\} \quad \text{relational-operator} \quad \left\{\begin{matrix}\text{identifier-2}\\ \text{literal-2}\end{matrix}\right\}$$

The first operand (identifier-1 or literal-1) is called the subject of the condition; the second operand (identifier-2 or literal-2) is called the object of the condition. The subject and the object may not both be literals.

The relational operator specifies the type of comparison to be made in a relation condition. The meaning of the relational operators is as shown in Figure 8-2. Notice that either the symbol or the word is allowable for the conditions greater than, less than, and equal to.

Relational Operator	Meaning
IS [NOT] <u>GREATER</u> THAN IS [NOT] >	Greater than or not greater than
IS [NOT] <u>LESS</u> THAN IS [NOT] <	Less than or not less than
IS [NOT] <u>EQUAL</u> TO† IS [NOT] =	Equal to or not equal to

†TO is optional on EQUAL TO relational operator.

Figure 8-2. Relational Operators and Their Meanings

COMPARISON OF NUMERIC OPERANDS

For operands whose category 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. Unsigned numeric operands are considered positive for purposes of comparison.

Figure 8-3 shows examples of the comparison of numeric operands.

Subject	Object	Result of Comparison
2 3 6 ^ 6 0	2 3 ^ 5 0 2 0	236.60 is greater than 23.5020
2 6 2	6 2 ^ 3 6	262 is greater than 62.36
2 0 0 0 ^ 0	2 0 0 0 ^ 2	2000.0 is less than 2000.2
2 ^ 0 -	0 0 ^ 0 +	-2.0 is less than +0.0
0 0 -	0 0 +	0 equals 0
4 5	0 2 3 9	45 is less than 0239
0 3 6	3 6	036 equals 36

Figure 8-3. Examples of Comparisons of Numeric Operands

COMPARISON OF NONNUMERIC OPERANDS

For nonnumeric operands, or one numeric and one nonnumeric operand, a comparison is made with respect to a specified collating sequence of characters. Refer to Appendix B for the collating sequence.

The size of an operand is the total number of characters in the operand. Numeric and nonnumeric operands may be compared only when their usage is the same, implicitly or explicitly.

The operands may be equal in size or unequal in size. For equal length operands, characters in corresponding character positions of the two operands are compared starting from the high-order end through the low-order end. If all pairs of characters compare equally through the last pair, the operands are considered equal when the low-order end is reached. 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 that is positioned higher in the collating sequence is considered to be the greater operand.

If the operands are of unequal size, comparison proceeds as though the shorter operand were extended on the right by a sufficient number of spaces to make the operands of equal size.

Figure 8-4 shows examples of the comparison of nonnumeric operands.

Subject	Object	Result of Comparison
C D 8 4 3	C D 8 4 3	CD843 is equal to CD843
B 3 4 0	8 3 4 0	B340 is greater than 8340
8 4 0	8 4 1	840 is less than 841
B C D E	T U V W	BCDE is less than TUVW
N O P Q R	B C D	NOPQR is greater than BCD
I A G M A K B A	I A G M A B A	IAGMAKBA is greater than IAGMABA
I J K M △ △ A K	I J K M △ △ A L	IJKM△△AK is less than IJKM△△AL
D E F	D E F G	DEF is less than DEFG
D E F	D E F △	DEF is equal to DEF△

Figure 8-4. Examples of Comparisons of Nonnumeric Operands

COMPARISONS INVOLVING INDEX-NAMES AND/OR INDEX DATA ITEMS

The comparison of two index-names is equivalent to the comparison of their corresponding occurrence numbers.

In the comparison of an index data item with an index-name or with another index data item, the actual values are compared without conversion.

The comparison of an index-name with a numeric item is permitted if the numeric item is an integer. The numeric integer is treated as an occurrence number. No other comparisons involving an index-name or index data item are allowed (see *Table Handling* in Chapter 9).

PERMISSIBLE COMPARISONS OF SUBJECT AND OBJECT OPERANDS

Table 8-1 lists all permissible comparisons of subject and object operands.

Following is a partial list of abbreviations and their explanations as used in Table 8-1.

- NN, comparison is made as described for nonnumeric operands
- NU, comparison is made as described for numeric operands

- IO, comparison is made as described for two index-names
- IV, comparison is made as described for index data items
- A blank in any column signifies an illegal comparison.

Table 8-1. Permissible Comparisons of Subject and Object Operands

Subject Operand \ Object Operand	GR	AL	AN	ANE	FC* NNL	ZR NL	ED	BI	ID	IN	IDI
Group (GR)	NN	NN	NN	NN	NN	NN	NN				
Alphabetic (AL)	NN	NN	NN	NN	NN	NN	NN				
Alphanumeric (AN)	NN	NN	NN	NN	NN	NN	NN				
Alphanumeric Edited (ANE)	NN	NN	NN	NN	NN	NN	NN				
Numeric Edited (NE)	NN	NN	NN	NN	NN	NN	NN				
Figurative Constant (FC)* Nonnumeric Literal (NNL)	NN	NN	NN	NN			NN				
Figurative Constant Zero (ZR) Numeric Literal (NL)	NN	NN	NN	NN			NU	NU	NU	IO**	
External Decimal (ED)	NN	NN	NN	NN	NN	NU	NU	NU	NU	IO**	
Binary (BI)						NU	NU	NU	NU	IO**	
Internal Decimal (ID)						NU	NU	NU	NU	IO**	
Index-Name (IN)						IO**	IO**	IO**	IO**	IO	IV
Index Data Item (IDI)										IV	IV

*Includes all figurative constants except zero.
**Valid only if the numeric item is an integer.

CONDITIONAL STATEMENTS

Conditional statements evaluate conditions which cause the object programs to choose between alternate paths of control. Only the IF statement is discussed in this section. Discussion of the other conditional statements is included as part of the description of the associated imperative statements.

IF STATEMENT

The IF statement causes a condition to be evaluated. The subsequent action of the object program depends on whether the value of the condition is true or false. The format of the IF statement is:

$$\underline{\text{IF}}\ \text{condition} \begin{Bmatrix} \text{statement-1} \\ \underline{\text{NEXT SENTENCE}} \end{Bmatrix} \underline{\text{ELSE}} \begin{Bmatrix} \text{statement-2} \\ \underline{\text{NEXT SENTENCE}} \end{Bmatrix}$$

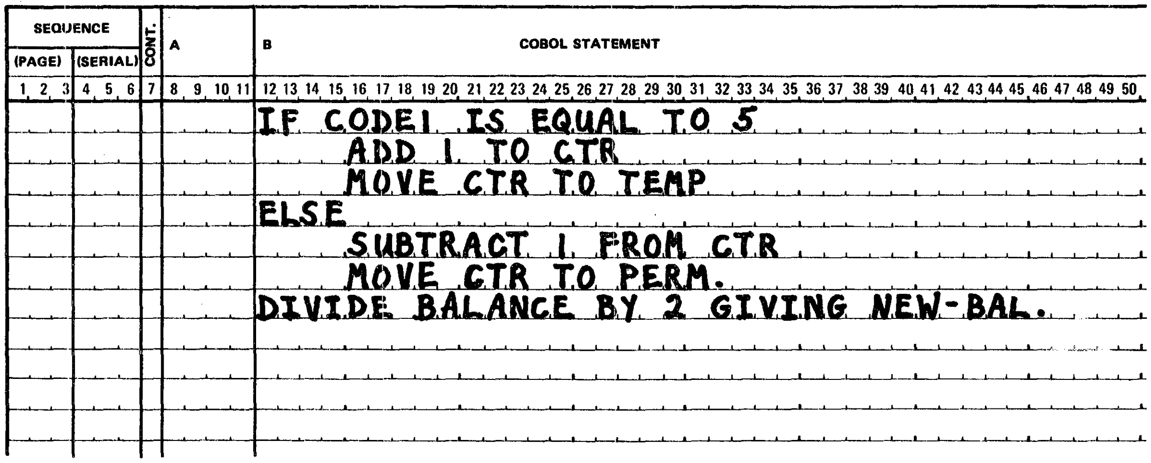
Statement-1 and statement-2 are imperative statements.

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:

- The condition is evaluated to be true or false.
- If true, the statements immediately following the condition (represented by statement-1) are executed; control then passes implicitly to the next sentence.
- If false, either the statements following the ELSE are executed or, if the ELSE clause is omitted, the next sentence is executed.

Example:



```
           IF CODE1 IS EQUAL TO 5
               ADD 1 TO CTR
               MOVE CTR TO TEMP
           ELSE
               SUBTRACT 1 FROM CTR
               MOVE CTR TO PERM.
           DIVIDE BALANCE BY 2 GIVING NEW-BAL.
```

In the program segment shown when CODE1 equals 5, the following steps are executed:

- Add 1 to CTR.
- Move CTR to TEMP.
- The next sentence, DIVIDE BALANCE BY 2 GIVING NEW-BAL, is executed.

If CODE1 does not equal 5, the following steps are executed.

- Subtract 1 from CTR.
- Move CTR to PERM.
- The next sentence, DIVIDE BALANCE BY 2 GIVING NEW-BAL, is executed

When an IF statement is executed and the NEXT SENTENCE option is present, control passes explicitly to the next sentence depending on the truth value of the condition and the placement of the NEXT SENTENCE clause in the statement.

IMPERATIVE STATEMENTS

An imperative statement unconditionally causes a specified function to occur at program execution time. Imperative statements discussed in this section are:

- Arithmetic
- Procedure Branching
- Data Manipulation
- Input/Output
- Subprogram Linkage

ARITHMETIC STATEMENTS

Arithmetic statements are used in computations and specify four operations:

- ADD
- SUBTRACT
- MULTIPLY
- DIVIDE

Each type of arithmetic statement includes options common to all four operations. They are: GIVING, ROUNDED, and SIZE ERROR. In addition, the data descriptions of the operands need not be the same; any necessary conversion and decimal point alignment is supplied throughout the calculation. The maximum size of each operand is 18 decimal digits.

GIVING OPTION

If the GIVING option is specified, the value of the identifier that follows the word GIVING is set equal to the calculated result of the arithmetic operation. This identifier, since it is not involved in the computation, may be a numeric edited item.

ROUNDED OPTION

If, after decimal point alignment, the fractional result of an arithmetic operation is greater than the number of places provided for the fraction by the resultant-identifier, truncation occurs to the size of the resultant-identifier. When rounding is requested, the absolute value of the resultant-identifier is increased by one (1) whenever the most significant digit of the excess is greater than or equal to five (5).

When the low-order integer positions in a resultant-identifier are represented by the character P in the picture for that resultant-identifier, rounding or truncation occurs relative to the right-most integer position for which storage is allocated.

SIZE ERROR OPTION

If, after decimal point alignment, the value of a result exceeds the largest value that can be contained in the associated resultant-identifier, a size error condition exists. Division by zero always causes a size error condition. The size error condition applies only to the final results of an arithmetic operation and does not apply to intermediate results, except in the MULTIPLY and DIVIDE statements, in which case the size error condition applies to the intermediate results as well. If the ROUNDED option is specified, rounding takes place before checking for size error. When such a size error condition occurs, the subsequent action depends on whether or not the SIZE ERROR option is specified.

If the SIZE ERROR option is not specified and a size error condition occurs, the value of the resultant-identifier may be unpredictable.

If the SIZE ERROR option is specified and a size error condition occurs, then the value of the resultant-identifier affected by the size error is not altered. After execution of this operation, the imperative-statement in the SIZE ERROR option is executed.

OVERLAPPING OPERANDS

When a sending and a receiving item in an arithmetic statement or MOVE statement share a part of their storage areas, the result of the execution of such a statement is undefined.

ADD STATEMENT

The ADD statement causes two or more numeric operands to be summed and the result to be stored. It has two formats which are:

Format 1:

$$\underline{\text{ADD}} \begin{Bmatrix} \text{identifier-1} \\ \text{literal-1} \end{Bmatrix} \begin{bmatrix} \text{identifier-2} \\ \text{literal-2} \end{bmatrix} \ldots$$

$$\underline{\text{TO}} \text{ identifier-m} [\underline{\text{ROUNDED}}]$$

$$[\text{ON } \underline{\text{SIZE ERROR}} \text{ imperative-statement}]$$

Format 2:

$$\underline{\text{ADD}} \begin{Bmatrix} \text{identifier-1} \\ \text{literal-1} \end{Bmatrix} \begin{bmatrix} \text{identifier-2} \\ \text{literal-2} \end{bmatrix} \begin{bmatrix} \text{identifier-3} \\ \text{literal-3} \end{bmatrix} \ldots$$

$$\underline{\text{GIVING}} \text{ identifier-m} [\underline{\text{ROUNDED}}] \; [\text{ON } \underline{\text{SIZE}}$$
$$\underline{\text{ERROR}} \text{ imperative-statement}]$$

In formats 1 and 2 each identifier must refer to an elementary numeric item, except that the identifier appearing to the right of the word GIVING may refer to a numeric edited data item.

Each literal must be a numeric literal.

The maximum size of each operand is eighteen (18) decimal digits. The composite of operands, which is that data item resulting from the superimposition of all operands, excluding the data item that follows the word GIVING, aligned on their decimal points, must not contain more than eighteen digits. There may not be more than 20 operands in a single ADD statement including the identifier m.

If format 1 is used, the value of the operands preceding the word TO are added together, then the sum is added to the current value in identifier-m, and the result is stored in the resultant identifier-m.

If format 2 is used, the values of the operands preceding the word GIVING are added together, then the sum is stored as the new value of resultant identifier-m.

The compiler incusres that enough places are carried so as not to lost significant digits during execution.

Figure 8-5 shows examples of the ADD statement.

SUBTRACT STATEMENT

The SUBTRACT statement is used to subtract one, or the sum of two or more numeric data items from an item, and sets the value of an item equal to the results. It has two formats which are:

Format 1:

$$\underline{\text{SUBTRACT}} \begin{Bmatrix} \text{literal-1} \\ \text{identifier-1} \end{Bmatrix} \begin{bmatrix} \text{literal-2} \\ \text{identifier-2} \end{bmatrix} \ldots$$

$$\underline{\text{FROM}} \text{ identifier-m } [\underline{\text{ROUNDED}}]$$

$$[\text{ON } \underline{\text{SIZE ERROR}} \text{ imperative-statement}]$$

Statement	Conditions Before Execution	Conditions After Execution	Result of Addition

ADD QUANT-1 TO QUANT-2	QUANT-1 = 2 8 1
	QUANT-2 = 0 3 4
		QUANT-1 = 2 8 1 *	QUANT-2 = 3 1 5

ADD 4.3 TO QUANT-3	LITERAL 4.3 = 4‸3 QUANT-3 = 1 5‸0	LITERAL 4.3 = 4‸3 *	QUANT-3 = 1 9‸3

ADD QUANT-1 QUANT-2 GIVING TOTAL	QUANT-1 = 1 3 8‸
	QUANT-2 2 4‸6
	TOTAL = 4 5 8‸8
		QUANT-1 = 1 3 8‸ *
		QUANT-2 = 2 4‸6 *
			TOTAL = 1 6 2‸6

ADD QUANT-1 QUANT-2 GIVING TOTAL ROUNDED	QUANT-1 = 1 3 8‸
	QUANT-2 = 2 4‸6
	TOTAL = 4 5 8
		QUANT-1 = 1 3 8‸ *
		QUANT-2 = 2 4‸6 *
			TOTAL = 1 6 3

ADD -47.1 to QUANT-3	LITERAL -47.1 = 4 7‸1 - QUANT-3 = 0 1 4 1‸2 5 +	LITERAL -47.1 = 4 7‸1 - *	QUANT-3 = 0 0 9 4‸1 5

ADD QUANT-4 QUANT-5 GIVING QUANT-6 ROUNDED ON SIZE ERROR GO TO ERR	QUANT-4 = 2 4 0
	QUANT-5 = 1 4 7 0‸2
	QUANT-6 = 8 8 5
		QUANT-4 = 2 4 0 *
		QUANT-5 = 1 4 7 0‸2
			QUANT-6 = 8 8 5 *
			Result exceeds capacity of QUANT-6. Statement at ERR is executed.

* = Unchanged

Figure 8-5. ADD Statement Examples

Format 2:

```
SUBTRACT {literal     } [literal-2   ] . . .
         {identifier-1} [identifier-2]

         FROM {literal-m   }
              {identifier-m}
                GIVING identifier-n [ROUNDED]

         [ON SIZE ERROR imperative-statement]
```

Each identifier must refer to a numeric elementary item except in format 2, where the identifier that appears to the right of the word GIVING may refer to a numeric edited data item. Each literal must be a numeric literal.

The maximum size of each operand is eighteen (18) decimal digits. The composite of operands, which is that data item resulting from the superimposition of all operands, excluding the identifier that follows the word GIVING, aligned on their decimal points, must not contain more than eighteen digits. There may not be more than 20 operands in a single SUBTRACT statement including identifier-m.

In format 1, all literals or identifiers preceding the word FROM are added together and this total is subtracted from identifier-m and the difference is stored as the new value of identifier-m.

In format 2, all literals or identifiers preceding the word FROM are added together, the sum is subtracted from literal-m or identifier-m and the result of the subtraction is stored as the new value of identifier-n.

The compiler insures that enough places are carried so as not to lose significant digits during execution.

Figure 8-6 shows examples of the SUBTRACT statement.

MULTIPLY STATEMENT

The MULTIPLY statement causes a numeric data item to be multiplied and sets the value of a data item equal to the result. It has two formats which are:

Format 1:

```
MULTIPLY {identifier-1} BY identifier-2 [ROUNDED]
         {literal-1   }
         [ON SIZE ERROR imperative-statement]
```

Format 2:

```
MULTIPLY {identifier-1} BY {identifier-2}
         {literal 1   }    {literal-2   }
         GIVING identifier-3 [ROUNDED]
         [ON SIZE ERROR imperative-statement]
```

Statement	Conditions Before Execution	Conditions After Execution	Result of Subtraction

SUBTRACT QUANT-1 FROM QUANT-2	QUANT-1 = 25‸10
	QUANT-2 = 52‸10
		QUANT-1 = 25‸10 *	QUANT-2 = 27‸00

SUBTRACT 267.4 FROM QUANT-3	Literal 267.4 = 267‸4 QUANT-3 = 367	Literal 267.4 = 267‸4 *	QUANT-3 = 099‸
SUBTRACT 267.4 FROM QUANT-3 ROUNDED	Literal 267.4 = 267‸4 QUANT-3 = 367	Literal 267.4 = 267‸4 *	QUANT-3 = 100‸

SUBTRACT QUANT-1 FROM QUANT-2 GIVING RESULT	QUANT-1 = 64
	QUANT-2 = 2830
	RESULT = 0100
		QUANT-1 = 64 *
		QUANT-2 = 2830 *
			RESULT = 2766

SUBTRACT -40 FROM QUANT-4 GIVING QUANT-5 ROUNDED	Literal -40 = 40-
	QUANT-4 = 264‸8-
	QUANT-5 = 381+
		Literal -40 = 40- *
		QUANT-4 = 264‸8- *
			QUANT-5 = 225-

* = Unchanged

Figure 8-6. SUBTRACT Statement Examples

Each identifier must refer to a numeric elementary item, except in format 2, where the identifier that appears to the right of the word GIVING may refer to a numeric edited data item.

Each literal must be a numeric literal.

The maximum size of each operand is eighteen (18) decimal digits.

When format 1 is used, the value of identifier-1 or literal-1 is multiplied by the value of identifier-2. The value of the multiplier (identifier-2) is replaced by this product.

When format 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.

Figure 8-7 shows examples of the MULTIPLY statement.

DIVIDE STATEMENT

The DIVIDE statement divides one numeric data item into another and sets the value of a data item equal to the results. It has two formats which are:

Format 1:

$$\underline{\text{DIVIDE}} \begin{Bmatrix}\text{identifier-1}\\ \text{literal-1}\end{Bmatrix} \underline{\text{INTO}}\text{ identifier-2 }[\underline{\text{ROUNDED}}]$$
$$[\text{ON }\underline{\text{SIZE ERROR}}\text{ imperative-statement}]$$

Format 2:

$$\underline{\text{DIVIDE}} \begin{Bmatrix}\text{identifier-1}\\ \text{literal-1}\end{Bmatrix} \begin{Bmatrix}\underline{\text{INTO}}\\ \underline{\text{BY}}\end{Bmatrix} \begin{Bmatrix}\text{identifier-2}\\ \text{literal-2}\end{Bmatrix}$$
$$\underline{\text{GIVING}}\text{ identifier-3}$$
$$[\underline{\text{ROUNDED}}]\ [\text{ON }\underline{\text{SIZE ERROR}}\text{ imperative-statement}]$$

Each identifier must refer to a numeric elementary item, except, in format 2 where the identifier that appears to the right of the word GIVING may refer to a numeric edited data item.

Each literal must be a numeric literal.

The maximum size of each operand is eighteen (18) decimal digits. The maximum size of the resulting quotient, after decimal point alignment is 18 decimal digits.

Division by zero always results in a size error condition.

When format 1 is used, the value of identifier-1 or literal-1 is divided into the value of identifier-2. The value of the dividend (identifier-2) is replaced by the quotient.

Statement	Conditions Before Execution	Conditions After Execution	Result of Modification

MULTIPLY BASE BY PERCENT GIVING DISCOUNT	BASE = 1 0 2 1
	PERCENT = 0‸3 0
	DISCOUNT = 9 0 1
		BASE = 1 0 2 1 *
		PERCENT = 0‸3 0 *
			DISCOUNT = 3 0 6

MULTIPLY QUANT-1 BY 4.88 GIVING REBATE ROUNDED	QUANT-1 = 4 0
	Literal 4.88 = 4‸8 8
	REBATE = 8 8 8
		QUANT-1 = 4 0 *
		Literal 4.88 = 4‸8 8 *
			REBATE = 1 9 5

MULTIPLY QUANT-2 BY 4.88 GIVING OVFLOW ON SIZE ERROR GO TO ERR	QUANT-2 = 4 0
	Literal 4.88 = 4‸8 8
	OVFLOW= 2 6‸0
		QUANT-2 = 4 0 *
		Literal 4.88 = 4‸8 8 *
			OVFLOW = 2 6‸0 *
			Result exceeds size of OVFLOW. Statement at ERR is executed.

MULTIPLY 20 BY QUANT-3	Literal 20 = 2 0 QUANT-3 = 0 0 3 2	Literal 20 = 2 0 *	QUANT-3 = 0 6 4 0

* = Unchanged

Figure 8-7. MULTIPLY Statement Examples

When format 2 is used, the value of identifier-1 or literal-1 is divided by or into identifier-2 or literal-2 and the result is stored in identifier-3.

Figure 8-8 shows examples of the DIVIDE statement.

PROCEDURE BRANCHING STATEMENTS

Statements, sentences, and paragraphs in the Procedure Division are ordinarily executed sequentially. The procedure branching statements (GO TO, ALTER, PERFORM, STOP, and EXIT) allow alterations in this sequence.

GO TO STATEMENT

The GO TO statement causes control to be transferred from one part of the Procedure Division to another. It has two formats which are:

```
GO TO procedure-name-1

GO TO procedure-name-1 [procedure-name-2] . . .
      procedure-name-n DEPENDING ON identifier
```

Each procedure-name is the name of a paragraph or section in the Procedure Division of the program.

Identifier is the name of a numeric elementary item described as an integer. When format 2 is used, there may not be more than 100 procedure-names specified.

Whenever a GO TO statement, represented for format 1, is executed, control is transferred to procedure-name-1 or to another procedure-name if the GO TO statement has been altered by an ALTER statement. (ALTER statement is described later in this section.)

When, in format 1, the GO TO statement is referred to by an ALTER statement, the following rules apply:

- The GO TO statement must have a paragraph-name.
- The GO TO statement must be the only statement in the paragraph.

A GO TO statement represented by format 2 causes control to be transferred to one of the specified procedures named procedure-name-1, procedure-name-2, etc., depending on the values of identifier being 1, 2, ..., n. If the value of identifier is anything other than the positive or unsigned integers 1, 2, ..., n, then the GO TO statement has no effect.

Example:

When the following GO TO statement is executed, control will be transferred to STATE-TAX, FED-TAX, or SOC-SEC depending on the value of DED-TYPE.

Statement	Conditions Before Execution	Conditions After Execution	Result of Division

DIVIDE 3 INTO QUANT-1 GIVING FRACTION	Literal 3 = 3
	QUANT-1 = 2 8
	FRACTION = 1 3‸3
		Literal 3 = 3 *
		QUANT-1 = 2 8 *
			FRACTION = 0 9‸3

DIVIDE QUANT-1 BY 3 GIVING FRACTION	QUANT-1 = 2 8
	Literal 3 = 3
	FRACTION = 1 3‸3
		Literal 3 = 3 *
		QUANT-1 = 2 8 *
			FRACTION = 0 9‸3

DIVIDE 3 INTO QUANT-1 GIVING RESULT	Literal 3 = 3
	QUANT-1 = 2 8
	RESULT = 0 0‸0 2
		Literal 3 = 3 *
		QUANT-1 = 2 8 *
			RESULT = 0 9‸3 3

DIVIDE QUANT-1 BY 3 GIVING RESULT ROUNDED	QUANT-1 = 2 8
	Literal 3 = 3
	RESULT = 1 3‸3
		QUANT-1 = 2 8 *
		Literal 3 = 3 *
			RESULT = 0 9‸3

DIVIDE 3 BY QUANT-1 GIVING RESULT ON SIZE ERROR GO TO ERR.	Literal 3 = 3
	QUANT-1 = 0‸0
	RESULT = 2 0
		Literal 3 = 3 *
		QUANT-1 = 0‸0 *
			RESULT = 2 0 *
			Result remains unchanged.
			Statement at ERR is executed.

* = Unchanged

Figure 8-8. DIVIDE Statement Examples

SEQUENCE		CONT.	A	B COBOL STATEMENT
(PAGE)	(SERIAL)			
1 2 3	4 5 6	7	8 9 10 11	12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
				GO TO STATE-TAX FED-TAX SOC-SEC
				DEPENDING ON DED-TYPE.

The following program segment is equivalent to the above GO TO statement.

```
IF DED-TYPE IS EQUAL TO 1 GO TO STATE-TAX.
IF DED-TYPE IS EQUAL TO 2 GO TO FED-TAX.
IF DED-TYPE IS EQUAL TO 3 GO TO SOC-SPEC ELSE NEXT SENTENCE.
.
.
.
```

ALTER STATEMENT

The ALTER statement modifies a predetermined sequence of operations. The format is as follows:

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

Procedure-name-1 is the name of a paragraph that contains only one sentence consisting of a GO TO statement without the DEPENDING ON option.

Procedure-name-2 is the name of a paragraph or section in the Procedure Division.

During execution of the object program, the ALTER statement modifies the GO TO statement in the paragraph named procedure-name-1, replacing the object of the GO TO by procedure-name-2.

Example:

```
CHANGE-GO.
   GO TO BYPASS-REPORT.
   .
   .
   .
PROCESS-TRANS.
   .
   .
   .
BYPASS-REPORT.
   .
   .
   .
   ALTER CHANGE-GO TO REPORT.
   .
   .
   .
REPORT.
   .
   .
   .
```

When CHANGE-GO is executed the first time, control is passed to BYPASS-REPORT. In BYPASS-REPORT, the ALTER statement when executed, modifies the GO TO statement in CHANGE-GO. When CHANGE-GO is executed the second time, control is transferred to REPORT.

A GO TO statement in a section whose priority is greater than or equal to 50 must not be referred to by an ALTER statement in a section with a different priority (see *Segmentation* in Section 9). All other uses of the ALTER statement are valid and are performed even if the GO TO to which the ALTER refers is in an overlayable fixed segment.

PERFORM STATEMENT

The PERFORM statement is used to depart from the normal sequence of execution, execute one or more procedures a specified number of times and return control to the normal sequence.

The PERFORM statement has two formats which are:

Format 1:

$\underline{\text{PERFORM}}$ procedure-name-1 [$\underline{\text{THRU}}$ procedure-name-2]

Format 2:

$\underline{\text{PERFORM}}$ procedure-name-1 [$\underline{\text{THRU}}$ procedure-name-2]

$\left\{\begin{array}{l}\text{identifier-1}\\ \text{integer-1}\end{array}\right\} \underline{\text{TIMES}}$

Each procedure-name is the name of a section or paragraph in the Procedure Division.

Identifier-1 represents a numeric elementary item with no positions to the right of the assumed decimal point, described in the Data Division.

When the PERFORM statement is executed, control is transferred to the first statement of the procedure named procedure-name-1. An automatic return to the statement following the PERFORM statement is established as follows:

- If procedure-name-1 is a paragraph-name and procedure-name-2 is not specified, then the return is after the last statement of procedure-name-1.

- If procedure-name-1 is a section-name and procedure-name-2 is not specified, then the return is after the last statement of the last paragraph in procedure-name-1.

- If procedure-name-2 is specified and it is a paragraph-name, then the return is after the last statement of the paragraph.

- If procedure-name-2 is specified and it is a section-name, then the return is after the last sentence of the last paragraph in this 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. If there are two or more direct paths to the return point, then procedure-name-2 may be the name of a paragraph consisting of the EXIT statement, to which all these paths must lead. The execution of the EXIT statement in this case, returns control to the statement following the PERFORM statement.

If control passes to these procedures by means other than a PERFORM statement, control passes through the last statement of the procedure to the following statement as if no PERFORM statement mentioned these procedures.

If a sequence of statements referred to by a PERFORM statement includes another PERFORM statement, the sequence of procedures associated with the included PERFORM must itself either be totally included in, or totally excluded from the logical sequence referred to by the first PERFORM. Thus, an active PERFORM statement, whose execution begins within the range of another active PERFORM statement, must not allow control to pass through the exit of the original PERFORM. Two or more active PERFORM statements may not have a common exit.

Format 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.

Format 2 is the TIMES option. When the TIMES option is used the procedures are performed the number of times specified by the initial value of identifier-1 or integer-1, for that execution. When the PERFORM statement is executed, the value of integer-1 must be positive.

If the initial value of identifier-1 is negative or zero, control passes immediately to the statement following the PERFORM statement. Following the execution of the procedures the specified number of times, control is transferred to the statement following the PERFORM statement.

During execution of the PERFORM statement, reference to identifier-1 will not alter the number of times the procedures are to be executed from that which was indicated by the initial value of identifier-1.

If integer-1 or identifier-1 is greater than 4 digits, significance may be lost. The maximum value is 2^{16}-1.

A PERFORM statement that appears in a section whose priority is less than the segment limit, can have within its range only the following:

- Sections each of which has a priority number less than 50.
- Sections wholly contained in a single segment whose priority number is greater than 49. (See *Segmentation* in Section 9.)

A PERFORM statement that appears in a section whose priority number is equal to or greater than the segment limit, can have within its range only the following:

- Sections each of which has the same priority number as that containing the PERFORM statement.
- Sections with a priority number that is less than the segment limit. (See *Segmentation* in Section 9.)

When a procedure-name in a segment with a priority number greater than 49 is referred to by a PERFORM statement contained in a segment with a different priority number, the segment referred to is made available in its initial state for each execution of the PERFORM statement. (See *Segmentation* in Section 9.)

Following are examples of the PERFORM statement.

Example 1: Basic PERFORM Without Procedure-Name-2

```
       A-1.
           MULTIPLY AMOUNT BY 3ØØ GIVING
           TOTAL-AMOUNT.
           PERFORM CALCULATE.
           ADD 1ØØ TO TOTAL.
           GO TO CONT-PROC.
       CALCULATE.
           ADD 1Ø TO TOTAL.
           MOVE TOTAL TO NEW-TOTAL.
           SUBTRACT TOTAL FROM NEW-TOTAL.
       MOVE-DATE.
           IF DATE IS EQUAL TO TODAY-DATE
           MOVE TODAY-DATE TO OUTPUT-DATE.
```

In example 1, the statement PERFORM CALCULATE executes the three statements contained in the paragraph CALCULATE. The instructions are executed in the following sequence:

```
MULTIPLY AMOUNT BY 300 GIVING TOTAL-AMOUNT.
ADD 10 TO TOTAL
MOVE TOTAL TO NEW-TOTAL.                  PERFORM
SUBTRACT TOTAL FROM NEW-TOTAL.            CALCULATE
ADD 100 TO TOTAL.
GO TO CONT-PROC.
 .
 .
 .
```

Example 2: Basic PERFORM With Procedure-Name-2

```
SEQUENCE        CONT.  A    B                    COBOL STATEMENT
(PAGE) (SERIAL)
1 2 3  4 5 6    7      8 9 10 11  12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50

       ABC.
           MOVE NEW-ACCT-NO TO ACCT-NO.
           PERFORM MOVEMENT THRU COMPUTATION.
           WRITE OUT-RECORD.
           GO TO OUTPUT-ROUTINE.
       TEST-EQUALITY.
           IF TCODE IS EQUAL TO 1 GO TO ROUTINE-1.
           IF TCODE IS EQUAL TO 2 GO TO ROUTINE-2.
           IF TCODE IS EQUAL TO 3 GO TO ROUTINE-3.
       MOVEMENT. MOVE BALANCE TO NEW-BALANCE.
           MOVE TCODE TO NEW-CODE.
       NEW-RECORD. MOVE 2 TO NEW-LL-CODE.
           MOVE DATE-1 TO NEW-DATE.
       COMPUTATION. ADD AMOUNT TO BALANCE.
           SUBTRACT 35 FROM LOWER-LIMIT.
       OUTPUT-ROUTINE. WRITE NEW-RECORD FROM
           OLD-AREA.
           ADD 1 TO COUNTER-1.

1 2 3  4 5 6    7      8 9 10 11  12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
```

In example 2 the statement PERFORM MOVEMENT THRU COMPUTATION executes six statements in the three paragraphs entitled MOVEMENT, NEW-RECORD, and COMPUTATION. The instructions are executed in the following sequence:

```
MOVE NEW-ACCT-NO TO ACCT-NO.
MOVE BALANCE TO NEW-BALANCE.
MOVE TCODE TO NEW-CODE.
MOVE 2 TO NEW-LL-CODE.
MOVE DATE-1 TO NEW-DATE.              PERFORM
ADD AMOUNT TO BALANCE.                MOVEMENT
                                      THRU
SUBTRACT 35 FROM LOWER-LIMIT.         COMPUTATION.
WRITE OUT-RECORD.
GO TO OUTPUT-ROUTINE.
```

The procedures referred to by the PERFORM statement are executed once. Control then passes to the statement following the PERFORM statement.

Example 3: Total Exclusion

In the following illustration of total exclusion, EXTRA-TOTAL is located completely outside the performed sequence ABC THRU XYZ.

```
    PERFORM-A. PERFORM ABC THRU XYZ.
    . . . . . . . . . . . . . . . . . .
ABC . . . . . . . . . . . . . . . . . .
    . . . . . . . . . . . . . . . . . .
    . . . . . . . . . . . . . . . . . .      Sequence performed by
    PERFORM EXTRA-TOTAL                      PERFORM ABC THRU XYZ
    . . . . . . . . . . . . . . . . . .
    . . . . . . . . . . . . . . . . . .
XYZ . . . . . . . . . . . . . . . . . .
    . . . . . . . . . . . . . . . . . .
EXTRA-TOTAL . . . . . . . . . . . . . .
    . . . . . . . . . . . . . . . . . .
MORE-INPUT  . . . . . . . . . . . . . .
```

Example 4: Total Inclusion

In the following illustration of total inclusion, COMPUTE-TAX is located completely inside the performed sequence DED-1 THRU END-1.

```
PERFORM-B. PERFORM DED-1 THRU END-1.
    . . . . . . . . . . . . . . .
DED-1. . . . . . . . . . . . . . .
    . . . . . . . . . . . . . . .
COMPUTE-TAX. . . . . . . . . . .
    . . . . . . . . . . . . . . .
MORE-CALCULATION. . . . . . . . .     Sequence performed by
    . . . . . . . . . . . . . . .     PERFORM DED-1 THRU
    PERFORM COMPUTE-TAX.              END-1.
    . . . . . . . . . . . . . . .
END-1. . . . . . . . . . . . . . .
    . . . . . . . . . . . . . . .
SUBROUTINE-A. . . . . . . . . . .
    . . . . . . . . . . . . . . .
```

Example 5: Overlapping Performed Sequences

```
PERFORM-C. PERFORM TEST-1 THRU ZERO-DIFFERENCE.
    . . . . . . . . . . . . . . .
PERFORM-D. PERFORM MOVE-BALANCE THRU BALANCE-TEST.
    . . . . . . . . . . . . . . .
TEST-1. . . . . . . . . .
    . . . . . . . . . .       Sequence performed by
ADDITION. . . . . . . .       PERFORM TEST-1 THRU
    . . . . . . . . . .       ZERO-DIFFERENCE.
MOVE-BALANCE. . . . . .
    . . . . . . . . . .
ZERO-DIFFERENCE. . . . .
    . . . . . . . . . .       Sequence performed by
OVER-DRAFT. . . . . . .       PERFORM MOVE-BALANCE
    . . . . . . . . . .       THRU BALANCE-TEST.
BALANCE-TEST. . . . . .
```

Example 6: PERFORMS With a Common End Point

```
PERFORM-1. PERFORM TEST-CODE THRU EXIT-POINT.
SENTENCE-1. . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . .
TEST-CODE.
     IF TCODE IS EQUAL TO 12 GO TO PATH-B.
     IF TCODE IS EQUAL TO 15 GO TO PATH-C.
     IF TCODE IS EQUAL TO 16 GO TO EXIT-POINT.
PATH-A. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . .
GO TO EXIT-POINT.
PATH-B. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . .
GO TO EXIT-POINT.
PATH-C. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . .
EXIT-POINT. EXIT.
```

In example 6 the original PERFORM statement named PERFORM-1 executes the set of procedures from TEST-CODE through EXIT-POINT. Within these procedures the testing of TCODE results in the execution of one of four different paths: PATH-A, PATH-B, PATH-C, and EXIT-POINT.

When TCODE equals 12, PATH-B is taken. At the end of PATH-B, a return is to be made to the statement (SENTENCE-1) following the original PERFORM statement. This is accomplished by the statement GO TO EXIT-POINT. EXIT-POINT is the paragraph-name of the EXIT associated with the original PERFORM statements.

When TCODE equals 15, PATH-C is taken. At the end of PATH-C, a return is to be made to the statement (SENTENCE-1) following the original PERFORM statement. Since the paragraph following the end of PATH-C is the EXIT-POINT paragraph containing the EXIT sentence, there is no need to use the sentence GO TO EXIT-POINT to return to the original PERFORM statement.

When TCODE equals 16, an immediate return is to be made to the statement (SENTENCE-1) following the original PERFORM statement. This is accomplished by the statement GO TO EXIT-POINT. EXIT-POINT is the paragraph-name of the EXIT associated with the original PERFORM statement.

When TCODE is not equal to 12, 15, or 16, then PATH-A is taken. At the end of PATH-A, a return is to be made to the statement (SENTENCE-1) following the original PERFORM statement. This is accomplished by the statement GO TO EXIT-POINT. EXIT-POINT is the paragraph name of the EXIT associated with the original PERFORM statement.

Example 7: PERFORM With TIMES Option

The procedure ADDITION-ROUTINE is executed three times.

```
        PERFORM ADDITION-ROUTINE 3 TIMES.
        . . . . . . . . . . . . . . . .
        . . . . . . . . . . . . . . . .
        . . . . . . . . . . . . . . . .
        . . . . . . . . . . . . . . . .
    ADDITION-ROUTINE . . . . . . . . . .
        . . . . . . . . . . . . . . . .
        . . . . . . . . . . . . . . . .
        . . . . . . . . . . . . . . . .
        . . . . . . . . . . . . . . . .
        . . . . . . . . . . . . . . . .
        . . . . . . . . . . . . . . . .
    TEST-ROUTINE . . . . . . . . . . . .
        . . . . . . . . . . . . . . . .
        . . . . . . . . . . . . . . . .
```

STOP STATEMENT

The STOP statement causes a permanent or temporary suspension of the execution of the object program. Its format is as follows:

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

The literal may be numeric or nonnumeric or may be any figurative constant. Signed numeric literals cause the development of a low-order sign overpunch.

If the RUN option is used, the execution of the object program is terminated, and control is returned to the system.

If the literal option is used, the literal is communicated to the operator. The program may be resumed only by operator intervention (key in a RETURN on the console). Continuation of the object program then begins with the execution of the next statement in sequence.

If a STOP statement with the RUN option appears in an imperative sentence, it must appear as the only or last statement in a sequence of imperative statements. All files should be closed before a STOP RUN statement is issued.

EXIT STATEMENT

The EXIT statement provides a common end point for a series of procedures. The format is as follows:

$\underline{\text{EXIT}}$ $[\underline{\text{PROGRAM}}]$.

The EXIT sentence must be preceded by a paragraph-name and must be the only sentence in the paragraph.

It is sometimes necessary to transfer control to the end point of a series of procedures. This is normally done by transferring control to the next paragraph or section, but in some cases this method does not produce the required result. For instance, the point to which control is to be transferred may be at the end of a range of procedures governed by a PERFORM. The EXIT statement is provided to enable a procedure-name to be associated with such a point.

If control reaches an EXIT paragraph and no associated PERFORM statement is active, control passes through the EXIT point to the first sentence of the next paragraph.

The EXIT statement with the PROGRAM option is discussed in *Subprogram Linkage Statements* in this section.

DATA MANIPULATION STATEMENTS

Movement and inspection of data are implicit in the functioning of the COBOL statements MOVE and EXAMINE.

MOVE STATEMENT

The MOVE statement transfers data, in accordance with the rules of editing, to one or more data areas. The format is as follows:

$$\underline{\text{MOVE}} \begin{Bmatrix} \text{identifier-1} \\ \text{literal} \end{Bmatrix} \underline{\text{TO}} \text{ identifier-2[,identifier-3]} \ldots$$

Identifier-1 and literal represent the sending area; identifier-2 and identifier-3 represent the receiving areas.

The data designated by the literal or identifier-1 is moved first to identifier-2, then to identifier-3. The notes referring to identifier-2 also apply to the other receiving areas. Any subscripting or indexing associated with the receiving area is evaluated immediately before the data is moved to the respective data item.

Any move in which the sending and receiving items are both elementary items is an elementary move. Every elementary item belongs to one of the following categories: numeric, alphabetic, alphanumeric, numeric edited, alphanumeric edited. These categories are described in the PICTURE clause. Numeric literals belong to the category numeric, and nonnumeric literals belong to the category alphanumeric. The figurative constant ZERO belongs to the category numeric. The figurative constant SPACE belongs to the category alphabetic. All other figurative constants belong to the category alphanumeric.

The following rules apply to an elementary move between these categories:

- The figurative constant SPACE, a numeric edited, alphanumeric edited, or alphabetic data item, must not be moved to a numeric or numeric edited data item.

- A numeric literal, the figurative constant ZERO, a numeric data item or a numeric edited data item must not be moved to an alphabetic item.

- A numeric literal, or a numeric data item whose implicit decimal point is not immediately to the right of the least significant digit, must not be moved to an alphanumeric or alphanumeric edited data item.

Any necessary conversion of data from one form of internal representation to another takes place during the legal elementary moves, along with any editing specified for the receiving data item. The following rules apply to legal elementary moves:

- When an alphanumeric edited, alphanumeric, or alphabetic item is a receiving item, justification and any necessary space-filling takes place as defined under the JUSTIFIED clause. If the size of the sending item is greater than the size of the receiving item, the excess characters are truncated after the receiving item is filled. If the sending item has an operational sign, the absolute value is used.

- When a numeric or numeric edited item is a receiving item, alignment by decimal point and any necessary zero-filling takes place, except where zeros are replaced because of editing requirements. If the receiving item has no operational sign, the absolute value of the sending item is used. If the sending item has more digits to the left or right of the decimal point than the receiving item can contain, the excess digits are truncated. When a data item described as alphanumeric is the sending item, it is moved as though it was described as an unsigned numeric integer item. If the sending item contains any nonnumeric characters, the results are undefined.

- When a receiving field is described as alphabetic and the sending data item contains any nonalphabetic characters, the results are undefined.

- When the sending and receiving operands of a MOVE statement share a part of their storage (that is, when the operands overlap), the result of the execution of such a statement is unpredictable.

An index data item cannot appear as an operand of a MOVE statement.

Any move that is not an elementary move is treated exactly as if it were an alphanumeric to alphanumeric elementary move, except that there is no conversion of data from one form of internal representation to another.

There are certain restrictions on elementary moves. These restrictions are listed in Table 8-2.

Figure 8-9 shows examples of the MOVE statement.

Table 8-2. Permissible Moves

Receiving Field / Source Field	GR	AL	AN	ED	BI	NE	ANE	ID
Group (GR)	Y	Y	Y	Y①	Y①	Y①	Y①	Y①
Alphabetic (AL)	Y	Y	Y	N	N	N	Y	N
Alphanumeric (AN)	Y	Y	Y	Y④	Y④	Y④	Y	Y④
External Decimal (ED)	Y①	N	Y②	Y	Y	Y	Y②	Y
Binary (BI)	Y①	N	Y	Y	Y	Y	Y①	Y
Numeric Edited (NE)	Y	N	Y	N	N	N	Y	N
Alphanumeric Edited (ANE)	Y	Y	Y	N	N	N	Y	N
ZERO (numeric or alphanumeric)	Y	N	Y	Y③	Y③	Y③	Y	Y③
SPACE (AL)	Y	Y	Y	N	N	N	Y	N
HIGH-VALUE, LOW-VALUE, QUOTE	Y	N	Y	N	N	N	Y	N
Numeric Literal	Y①	N	Y②	Y	Y	Y	Y②	Y
Nonnumeric Literal	Y	Y	Y	Y⑤	Y⑤	Y⑤	Y	Y⑤
Internal Decimal (ID)	Y①	N	Y②	Y	Y	Y	Y②	Y

Y – YES
N – NO

①Move without conversion (like AN to AN).
②Only if the decimal point is at the right of the least significant digit.
③Numeric move.
④The alphanumeric field is treated as an ED (integer) field.
⑤The literal must consist only of numeric characters and is treated as an ED integer field.

MOVE Statement	Item	Item PICTURE	Value Before Execution	Value After Execution
MOVE ZEROS TO FIELD-A	FIELD-A	9999	0123	0000
MOVE FIELD-1 TO FIELD-2	FIELD-1	XXX	ABC	ABC
	FIELD-2	XXXX	XYWK	ABCΔ
MOVE FIELD-3 TO FIELD-4	FIELD-3	XXX	ABC	ABC
	FIELD-4	XX	XY	AB
MOVE '123' TO FIELD-5	FIELD-5	XXXX	ABCD	123Δ
MOVE ACCOUNT-NO TO PR-ACCT-NO.	ACCOUNT-NO	XXXX	A123	A133
	PR-ACCT-NO	XBXXX	AΔCDE	AΔ123
MOVE 125.7 TO DOLLARS	DOLLARS	9999V99	1234‸66	012570
MOVE AMOUNT TO PR-AMOUNT	AMOUNT	9999V99	1258‸39	125839
	PR-AMOUNT	$9,999.99	$3333.33	$1,258.39
MOVE AMT-1 TO PR-AMOUNT-1	AMT-1	9999V99	0000‸03	000003
	PR-AMOUNT-1	$$,$$$.99	ΔΔΔ$23.19	ΔΔΔΔΔ$.03
MOVE FIELD-6 TO FIELD-7	FIELD-6	999	157	157
	FIELD-7	XXXX	AB24	157Δ

Figure 8-9. MOVE Statement Examples

EXAMINE STATEMENT

The EXAMINE statement replaces and/or counts the number of occurrences of a given character in a data item. The format is as follows:

```
        EXAMINE identifier
      ⎧                 ⎧UNTIL FIRST⎫                          ⎫
      ⎪   TALLYING      ⎨ALL        ⎬ literal-1                ⎪
      ⎪                 ⎩LEADING    ⎭                          ⎪
      ⎨                 [REPLACING BY literal-2]               ⎬
      ⎪                 ⎧ALL        ⎫                          ⎪
      ⎪   REPLACING     ⎨LEADING    ⎬ literal-3 BY literal-4   ⎪
      ⎪                 ⎪FIRST      ⎪                          ⎪
      ⎩                 ⎩UNTIL FIRST⎭                          ⎭
```

The description of the identifier must be such that usage is displayed either explicitly or implicitly.

Each literal must consist of a single character belonging to a class consistent with that of identifier. If identifier is numeric, each literal must be an unsigned integer or the figurative constant ZERO.

Examination proceeds as follows:

- For nonnumeric data items, examination starts at the left-most characters and proceeds to the right. Each character in the data item specified by the identifier is examined in turn.
- If a data item referred to by the EXAMINE statement is numeric, it may consist of numeric characters and may possess an operational sign. Examination starts at the left-most character (excluding the sign) and proceeds to the right. Each character except the sign is examined in turn. Regardless of where the sign is physically located, it is completely ignored by the EXAMINE statement.

The TALLYING option creates an integral count which replaces the value of a special software register called TALLY whose implicit description is that of an unsigned integer of five digits. The count represents the number of:

- Occurrences of literal-1 when the ALL option is used.
- Occurrences of literal-1 prior to encountering a character other than literal-1 when the LEADING option is used.
- Characters not equal to literal-1 encountered before the first occurrence of literal-1 when the UNTIL FIRST option is used.

When either of the REPLACING options is used, the replacement rules are as follows:

- When the ALL option is used, then literal-2 or literal-4 is substituted for each occurrence of literal-1 or literal-3.
- When the LEADING option is used, the substitution of literal-2 or literal-4 terminates as soon as a character other than literal-1 or literal-3 or the right-hand boundary of the data item is encountered.
- When the UNTIL FIRST option is used, the substitution of literal-2 or literal-4 terminates as soon as literal-1 or literal-3 or the right-hand boundary of the data item is encountered.
- When the first option is used, the first occurrence of literal-3 is replaced by literal-4.

Specific EXAMINE statements showing the effect of each statement on the associated data item and TALLY are shown in Figure 8-10.

EXAMINE Statement	ITEM-1 (Before)	Data (After)	Resulting Value of TALLY
EXAMINE ITEM-1 TALLYING ALL 0	101010	101010	3
EXAMINE ITEM-1 TALLYING ALL 1 REPLACING BY 0	101010	000000	3
EXAMINE ITEM-1 REPLACING LEADING "*" BY SPACE	**7000	ΔΔ7000	NA
EXAMINE ITEM-1 REPLACING FIRST "*" BY "$"	**1.94	$*1.94	NA

Figure 8-10. Examples of Data Examination

INPUT-OUTPUT STATEMENTS

The flow of data through the computer is governed by the operating system. The COBOL statements discussed in this section are used to initiate the flow of data to and from files stored on external media, and to govern low-volume information that is to be obtained from or sent to input/output devices such as a card reader or console typewriter.

The Operating System is a physical record processing system. That is, the unit of data made available by a READ or passed along by a WRITE is a physical record. However, the COBOL user need be concerned only with the use of individual logical records. The operating system provides such operations as the movement of data into buffers and/or internal storage, validity checking, error correction (where feasible), unblocking and blocking, and volume switching procedures.

Discussions in this section use the terms volume and reel. The term volume applies to all input/output devices. The term reel applies only to tape devices. Treatment of mass storage devices in the sequential access mode is logically equivalent to the treatment of tape files.

Input-output statements are: OPEN, SEEK, START, READ, WRITE, REWRITE, DELETE, ACCEPT, DISPLAY, CLOSE.

OPEN STATEMENT

The OPEN statement initiates processing of files. It performs checking and/or writing of labels and other input-output operations.

The format is as follows:

$$\underline{\text{OPEN}} \left\{ \begin{array}{l} \underline{\text{INPUT}} \text{ file-name [WITH } \underline{\text{NO REWIND}}] \\ \underline{\text{OUTPUT}} \text{ file-name [WITH } \underline{\text{NO REWIND}}] \\ \underline{\text{I-O}} \text{ file-name} \end{array} \right\} \ldots$$

At least one of the options: INPUT, OUTPUT, or I-O must be specified.

The I-O option pertains only to mass storage files.

The file-name must be defined by a file description entry in the Data Division.

An OPEN statement must be specified for all files. The OPEN statement for a file must be executed prior to the first READ, SEEK, WRITE, REWRITE, START, or DELETE statement for that file. A second OPEN statement for a file cannot be executed prior to the execution of a CLOSE statement for that file. The OPEN statement does not obtain or release the first data record. A READ or WRITE statement must be executed to accomplish this.

The NO REWIND option does not apply to unit record or disc files.

For tape files, the following rules apply:

1. When the NO REWIND option is specified, execution of the OPEN statement does not cause the file to be repositioned. The file must have been previously positioned at its beginning.

2. Without the NO REWIND option specified, execution of the OPEN statement causes the file to be positioned at its beginning.

The I-O option permits the opening of a mass storage file for both input and output operations. Since this option implies the existence of the file, it cannot be used if the mass storage file is being initially created.

A file may be opened as INPUT, OUTPUT or I-O in any order (with intervening CLOSE statements without the UNIT or REEL option).

When an indexed file in the sequential access mode is opened for I-O, the primary key-value of the first record in the file will be placed in the FORWARD KEY field, if the FORWARD KEY clause is specified.

SEEK STATEMENT

The SEEK statement initiates the accessing of a mass storage data record for subsequent reading or writing. The format is as follows:

SEEK file-name RECORD

A SEEK statement pertains only to relative files in the random access mode and may be executed prior to the execution of a READ or WRITE statement.

The file-name must be defined by a file description entry in the Data Division.

The SEEK statement uses the contents of the data-name in the ACTUAL KEY clause for the location of the record to be accessed. At the time of execution the contents of the ACTUAL KEY data item for the particular mass storage file is checked for validity. If the key is invalid, the next READ or WRITE statement on the associated file will give control to the imperative-statement in the INVALID KEY option.

Two SEEK statements for the same relative file may logically follow each other. Any validity check associated with the first SEEK statement is negated by the execution of the second SEEK statement.

If the contents of the ACTUAL KEY are altered between the SEEK statement and the subsequent READ or WRITE statement, any validity check associated with the SEEK statement is negated, and the READ or WRITE statement is processed as if no SEEK statement preceded it.

START STATEMENT

The START statement initiates processing of a segment of a sequentially accessed indexed file at a specified key. The format is as follows:

START file-name [INVALID KEY imperative-statement]

The key at which processing is to begin must be stored in the data-name specified in the FORWARD KEY clause before executing the START statement.

Normally, an indexed file in the sequential mode is processed sequentially from the first record to the last or until the file is closed. If processing is to begin or continue at other than the first or next sequential record, a START statement must be executed prior to the READ for the record desired. Processing will then continue sequentially until a START statement or a CLOSE statement is executed or until the end-of-file is reached.

If processing is to begin at the first record, a START statement is not required before the first READ.

If the INVALID KEY is specified, the contents of the FORWARD KEY field must correspond to the key value of a record in the file. If a record with that key value does not exist, control is passed to the imperative statement following INVALID KEY. If a record with that key value does exist, control is passed to the next sentence. In both cases, FORWARD KEY will contain the original key value upon return.

If the INVALID KEY option is not specified, two conditions may exist:

- The key value contained in FORWARD KEY matches the key value of an existing record. In this case, control is passed to the next statement, and FORWARD KEY will contain the original key value.
- A matching record is not found in the file. In this case control is passed to the next statement, and FORWARD KEY will contain the key value of the next sequential record with a key value higher than the key value specified in FORWARD KEY.

READ STATEMENT

For sequential file processing, the READ statement makes available the next logical record from an input file and allows performance of a specified imperative statement when end of file is detected.

For random file processing, the READ statement makes available a specific record from a mass storage file and allows performance of a specified imperative statement if the contents of the associated ACTUAL KEY data item are found to be invalid.

For sequential processing of indexed files, the READ statement places the primary key value of the next record in sequence into the FORWARD KEY field when a FORWARD KEY is defined for the file.

The format of the READ statement is as follows:

$$\underline{\text{READ}}\ \text{file-name RECORD}\ [\underline{\text{INTO}}\ \text{identifier}] \left\{ \begin{array}{l} \text{AT}\ \underline{\text{END}} \\ \underline{\text{INVALID}}\ \text{KEY} \end{array} \right\}$$
imperative-statement

An OPEN statement must be executed for the file prior to the execution of the first READ for that file.

When a READ statement is executed, the next logical record in the named file becomes accessible in the input area defined by the associated record description entry.

The record remains in the input area until the next input/output statement for that file is executed. No reference can be made by any statement in the Procedure Division to information that is not actually present in the current record. Thus, it is not permissible to refer to the nth occurrence of data that appears fewer than n times. If such a reference is made, no assumption should be made about the results in the object program.

When a file consists of more than one type of logical record, these records automatically share the same storage area; this is equivalent to an implicit redefinition of the area. Only the information that is present in the current record is accessible.

FILE-NAME must be defined by a file description entry in the Data Division.

INTO IDENTIFIER OPTION makes the READ statement equivalent to a READ statement plus a MOVE statement. Identifier must be the name of a Working Storage Section or Linkage Section entry, or an output record of a previously opened file. When this option is used, the current record becomes available in the input area, as well as in the area specified by the identifier. Data will be moved into identifier in accordance with the COBOL rules for moving group items.

AT END OPTION must be specified for all files in the sequential access mode. If, during the execution of a READ statement, the logical end of the file is reached, control is passed to the imperative-statement specified in the AT END phrase. After execution of the imperative-statement associated with the AT END phrase, a READ statement for that file must not be given without prior execution of a CLOSE statement, followed by an OPEN statement for that file.

If, during the processing of a multivolume file in the sequential access mode, end-of-volume is recognized on a READ, the following operations are carried out:

- The standard ending volume label procedure.
- A volume switch.
- The standard beginning volume label procedure.
- The first data record of the new volume is made available.

INVALID KEY OPTION: If ACCESS IS RANDOM is specified for the file, the contents of the ACTUAL KEY for the file must be set to the desired value before the execution of the READ statement.

For a randomly accessed file, the READ statement implicitly performs the functions of the SEEK statement, unless a SEEK statement for the file has been executed prior to the READ statement.

The INVALID KEY option must be specified for files in the random access mode. The imperative-statement following INVALID KEY is executed when the contents of the ACTUAL KEY field are invalid.

The key is considered invalid under the following conditions:

1. For a relative file that is accessed randomly, when the value specified by ACTUAL KEY is outside the limits specified in the FILE-LIMITS clause.

2. For an indexed file that is accessed randomly, when no record exists whose primary key-value matches the contents of the ACTUAL KEY field.

WRITE STATEMENT

The WRITE statement releases a logical record for an output file. It can also be used for vertical positioning of a print file. For mass storage files, the WRITE statement also allows the performance of a specified imperative statement if the file-limit is exceeded. For randomly accessed files the WRITE statement also allows the performance of a specified imperative statement if the contents of the associated ACTUAL KEY data item are found to be invalid.

The WRITE statement has two formats which are:

Format 1: $\underline{\text{WRITE}}$ record-name [$\underline{\text{FROM}}$ identifier-1]

$$\left[\begin{Bmatrix}\underline{\text{BEFORE}}\\ \underline{\text{AFTER}}\end{Bmatrix}\right]\text{ADVANCING}\begin{Bmatrix}\text{identifier-1 LINES}\\ \text{integer LINES}\end{Bmatrix}$$

Format 2: $\underline{\text{WRITE}}$ record-name [$\underline{\text{FROM}}$ identifier-1]

$\underline{\text{INVALID}}$ KEY imperative-statement

An OPEN statement for a file must be executed prior to executing the first WRITE statement for that file.

For files in both the sequential and random access modes, the logical record released is no longer available after the WRITE statement is executed.

RECORD NAME is the name of a logical record in the File Section of the Data Division.

When the FROM option is written, the WRITE statement is equivalent to the statement MOVE identifier-1 to record-name followed by the statement WRITE record-name. Moving takes place according to the COBOL rules for the MOVE statement. Identifier-1 should be defined in the Working Storage Section, the Linkage Section, or in another FD.

Format 1 is used only with standard sequential files.

If the sequential file is a printer or card punch the user must reserve the first character in each logical record for the control character. It is the user's responsibility, unless the ADVANCING option is specified, to insure the correct control character is set in the first character before the WRITE is issued. Refer to Appendix D for a list of ANS control characters.

The ANS control character will be converted by the system into the correct device dependent control function.

It is the user's responsibility to see that the appropriate channels are punched on the carriage control tape.

When the ADVANCING option is specified, the compiler will generate the appropriate control character in the first character position of the record. This control character will be one of the ANS Standard Control Characters.

When identifier-2 is used in the ADVANCING option, it must be the name of an unsigned numeric elementary item described as an integer. The maximum size of the item is two digits thus allowing a value range from 0 to 99.

When identifier-2 is specified, the printer page is advanced the number of lines equal to the value in the identifier.

When integer is used in the ADVANCING option, it must be an unsigned integer from 0 to 99. When integer is specified, the printer page is advanced the number of lines equal to the value of the integer.

If the BEFORE ADVANCING option is used, the record is written before the printer page is advanced according to the preceding rules.

If the AFTER ADVANCING option is used, the record is written after the printer page is advanced according to the preceding rules.

NOTE

> DISPLAY and WRITE AFTER ADVANCING statements cause the printer to space before printing. However, a WRITE BEFORE ADVANCING statement causes the printer to space after printing. Therefore, it is possible that mixed DISPLAY, WRITE AFTER ADVANCING and WRITE BEFORE ADVANCING within the same program may cause overprinting.

Format 2 is used for randomly or sequentially accessed mass storage files.

If ACCESS IS RANDOM is specified for the file, the contents of the ACTUAL KEY field for the file must be set to the desired value before the execution of a WRITE statement.

The INVALID KEY phrase must be specified for a file that resides on a mass storage device. Control is passed to the imperative-statement following INVALID KEY when the following conditions exist:

1. For a mass storage file in the sequential access mode opened as OUTPUT, when an attempt is made to write beyond the limit of the file.

2. For a relative file opened as I-O or OUTPUT, if access is random and a record is being added to the file, when the record address specified in the ACTUAL KEY field is outside the limits of the file, as specified by the FILE-LIMITS clause.

3. For an indexed file in the sequential access mode, opened as OUTPUT, and either one of the following conditions occurs:

   a. The contents of the ACTUAL KEY field are not in ascending order when compared with the contents of the ACTUAL KEY field of the preceding record.

   b. The contents of the ACTUAL KEY field duplicate that of the preceding record.

4. For an indexed file in the sequential access mode, opened as I-O, and a record is added to the file: if the contents of the ACTUAL KEY field associated with the record to be added is less than or equal to the key-value of the last record read, or greater than or equal to the key-value of the next record in sequence.

5. For an indexed file in the random access mode, opened as I-O, and a record is being added to the file: if the contents of the ACTUAL KEY field associated with the record to be added duplicates the contents of a primary-key field already in the file.

For randomly accessed files, the WRITE statement performs the function of a SEEK statement, unless a SEEK statement for this record is executed prior to the WRITE statement.

After the recognition of an end-of-volume on a multivolume OUTPUT or I-O file in the sequential access mode, the WRITE statement performs the following operations:

1. The standard ending volume label procedure.

2. A volume switch.

3. The standard beginning volume label procedure.

REWRITE STATEMENT

The function of the REWRITE statement is to replace a logical record in an indexed file with a specified record, if the contents of the associated ACTUAL KEY is found to be valid. The format is as follows:

```
REWRITE record-name [FROM identifier]
        INVALID KEY imperative-statement
```

The READ statement for a file must be executed before a REWRITE statement for the file can be executed. A REWRITE statement can be executed only for indexed files opened as I-O.

The ACTUAL KEY must be set to the desired value prior to the execution of the REWRITE statement.

The record-name is the name of a logical record description in the File Section of the Data Division.

When the FROM option is used, the REWRITE statement is equivalent to the statement MOVE identifier TO record-name followed by the statement REWRITE record-name. Identifier should be defined in the Working-Storage Section, Linkage Section, or in another FD.

For an indexed file that is accessed randomly, control is passed to the imperative-statement following INVALID KEY when the contents of the ACTUAL KEY is different from the contents used during execution of the preceding READ statement.

For an indexed file that is accessed sequentially, control is passed to the imperative-statement following INVALID KEY when the contents of the ACTUAL KEY do not match the primary key value associated with the record obtained by the preceding READ. (Specification of the FORWARD KEY clause may be used to obtain the primary key value, if the primary key field is not contained within the record itself.)

NOTE

> A WRITE statement executed on a relative file in the sequential access mode assumes the meaning of a REWRITE statement when the file is opened as I-O, and the WRITE is the next input/output operation following a READ.

DELETE STATEMENT

The function of the DELETE statement is to delete a logical record from an indexed file opened for I-O. The data record and its associated key values are removed. The format is as follows:

```
DELETE file-name RECORD
        INVALID KEY imperative-statement
```

The contents of the ACTUAL KEY must be set to the desired value before the execution of the DELETE statement.

If ACCESS IS SEQUENTIAL is specified for the file, the execution of the DELETE statement must be preceded by the execution of a READ statement for that file.

For an indexed file that is accessed randomly, control is passed to the imperative-statement following INVALID KEY when no record exists with a primary key value matching the contents of the ACTUAL KEY field.

For an indexed file that is accessed sequentially, if the contents of the ACTUAL KEY do not match the primary key value associated with the record obtained by the preceding READ, control is passed to the imperative-statement following INVALID KEY.

(If the primary key field is not contained within the record itself, specification of the FORWARD KEY clause may be used to obtain the primary key value of a record retrieved sequentially.)

ACCEPT STATEMENT

The ACCEPT statement causes low volume data to be transferred from an appropriate hardware device. The format is as follows:

```
     ACCEPT identifier [FROM mnemonic-name]
```

Identifier may be either a fixed-length group item or an elementary alphabetic, alphanumeric, or external decimal item. The data is read and the appropriate number of characters is moved into the area reserved for identifier. No editing or error checking of the incoming data is done.

Mnemonic-name may assume either the meaning SYSIN or CONSOLE. Mnemonic-name must be specified in the SPECIAL NAMES paragraph of the Environment Division. If the FROM option is not specified, CONSOLE is assumed.

For an ACCEPT with the FROM mnemonic-name for CONSOLE or if the FROM option is not specified, the following actions are taken:

- A system generated message code is automatically displayed followed by the literal "AWAITING REPLY".
- Program execution is suspended. When a console input message, preceded by the same message code as in point 1 above, is identified by the control program, execution of the ACCEPT statement is resumed and the message is moved to the specified identifier and left justified, regardless of the PICTURE. If the field is not filled, the low order positions may contain invalid data. Depressing RETURN from the console will terminate the ACCEPT statement.

Identifier must not exceed 100 character positions when accepting from the CONSOLE.

If mnemonic-name is associated with SYSIN, an input record size is the size of the //PAR Control Language statement minus 14 bytes. The maximum //PAR statement is assumed 128 bytes. The following three examples of the //PAR statement reflect an 80-column card, 96-column card and 128 byte terminal entry. (The //PAR statement is defined in the **MRX/OS Control Language Reference** manual.)

```
1——6 7——————————72  73——80
//PAR△ data - up to 66 bytes   sequence no.

1——6 7——————————88  89——96
//PAR△ data - up to 82 bytes   sequence no.

1——6 7——————————120 121——128
//PAR△ data - up to 114 bytes  sequence no.
```

The size of the input record is the data only. The first 6 bytes (//PAR△) and last 8 bytes, sequence number are dropped.

If the size of the accepting data item is less than the input data record, the input data record will be truncated on the right. If the size of the accepting data item is greater than the input data record size, as many input records as necessary are read until the storage area allocated to the data item is filled. If the accepting data item is greater than one input data record, but is not an exact multiple of the input data record size, the remainder of the last input record is not accessible.

DISPLAY STATEMENT

The DISPLAY statement causes low volume data to be transferred to an appropriate hardware device. The format is as follows:

$$\underline{\text{DISPLAY}} \begin{Bmatrix} \text{literal-1} \\ \text{identifier-1} \end{Bmatrix} \begin{bmatrix} \text{literal-2} \\ \text{identifier-2} \end{bmatrix} \ldots$$

$$[\underline{\text{UPON}}\ \text{mnemonic-name}]$$

Mnemonic-name must be specified in the SPECIAL NAMES paragraph of the Environment Division. Menmonic-name may be associated only with the reserved words CONSOLE and SYSOUT. When the UPON option is omitted, CONSOLE is assumed.

Identifiers described as USAGE COMPUTATIONAL, and COMPUTATIONAL-3, are converted automatically to external format, as follows:

- Internal decimal and binary items are converted to external decimal. Signed values cause a low-order sign overpunch to be developed. For example, if three internal-decimal items have values of -34, +34, and 34, they are displayed as 3M, 3D, and 34, respectively.

- No other data items require conversion.

If a figurative constant is specified as one of the operands, only a single occurrence of the figurative constant is displayed.

When a DISPLAY statement contains more than one operand, the data contained in the first operand is stored as the first set of characters, and so on, until the output record is filled. This operation continues until all information is displayed. Data contained in an operand may extend into subsequent records.

The DISPLAY and WRITE AFTER ADVANCING statements all cause the printer to space before printing. However, a WRITE BEFORE ADVANCING statement causes the printer to space after printing. Therefore, it is possible that mixed DISPLAY statements, WRITE AFTER ADVANCING statements within the same program may cause overprinting.

A maximum logical record size is assumed for each device. For CONSOLE (the system logical console device), the maximum is 100 characters. For SYSOUT (the system logical output device), the maximum is 120 characters.

If the total character count of all operands is less than the maximum, the remaining character positions are padded with blanks. If the count exceeds the maximum size, operands are continued in the next record. As many records as necessary are written to display all the operands specified. Those operands pending at the time of the break are split between lines if necessary.

CLOSE STATEMENT

The CLOSE statement terminates the processing of reels, units, and files, with optional rewind and/or lock where applicable.

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

The file must have been previously opened before a CLOSE statement can be executed.

The statement applies to the following categories of input and output files:

- Unit record volume. A file allocated on a medium for which rewinding, units, and reels have no meaning.
- Sequential single-volume tape. A sequential file that is contained entirely on one reel.
- Sequential multivolume tape. A sequential file that may be contained on more than one reel.
- Sequential single-volume disc. A sequential file that is contained entirely on one unit.
- Sequential multivolume disc. A sequential file that may be contained on more than one unit.
- Random. A random access file that is contained on one or more mass storage units.

The results of executing each close option for each type of file are summarized in Figure 8-11. Definitions of the symbols used in the figure are given below. Where the definition depends on whether the file is an input or output file, alternative definitions are given. Otherwise, a definition applies to input, output, and input-output files.

Following are the definitions of the symbols used in Figure 8-11:

- S – Standard close non-tape file

  System closing procedures are performed.

- T – Standard close tape file

  Files Opened as INPUT: If the file is positioned at its end and there is an ending label record, the standard ending label record procedures are performed. System closing procedures are then performed.

  If the file is positioned at its end and there is no ending label record, system closing procedures are performed.

  If the file is not positioned at its end, system closing procedures are performed.

  Files Opened at OUTPUT: If an ending label record has been described for the file, it is constructed and written on the output tape. System closing procedures are then performed.

- R – Rewind

  The current volume is positioned at its beginning.

- A – Previous volumes unaffected

  All volumes prior to the current volume have been processed according to standard volume switch procedures except those volumes controlled by a prior CLOSE REEL/UNIT statement.

- B – No rewind

  The current volume is left in its current position.

- E – Standard file lock

  The compiler ensures that this file cannot be opened again during this execution of the object program.

- F – Standard close reel

  Files Opened as INPUT: The following operations are performed:

a. A volume switch.

b. The standard beginning volume label procedure.

c. Makes the next data record on the new volume (reel) available to be read.

Files Opened as OUTPUT: The following operations are performed:

a. The standard ending volume label procedure.

b. A volume switch.

c. The standard beginning volume label procedure.

- C – Standard close unit

  Files Opened as INPUT or I-O: The volume is switched and the first data record on the new volume is made available.

  Files Opened as OUTPUT: A volume switch is performed.

- X – Illegal

  This is an illegal combination of a close option and a file type.

(CLOSE REEL/UNIT WITH LOCK/NO REWIND has no meaning in this system and is processed as a CLOSE REEL.)

File Type / CLOSE Option	Unit Record	Sequential Tape Single Reel	Sequential Tape, Multi-Reel	Sequential Disk Single Unit	Sequential Disk, Multi-Unit	Random
CLOSE	S	T, R	T, R, A	S	S, A	S
CLOSE WITH NO REWIND	X	T, B	T, B	X	X	X
CLOSE WITH LOCK	S, E	T, R, E	T, R, A, E	S, E	S, A, E	S, E
CLOSE REEL	X	X	F, R	X	X	X
CLOSE UNIT	X	X	X	X	C	X
CLOSE UNIT WITH LOCK	X	X	X	X	C	X
CLOSE REEL WITH NO REWIND	X	X	F, R	X	X	X
CLOSE REEL WITH LOCK	X	X	F, R	X	X	X

Figure 8-11. CLOSE Option and File Type Comparison

SUBPROGRAM LINKAGE STATEMENTS

Subprogram linkage statements are special statements that permit communication between object programs. These statements are CALL and EXIT PROGRAM.

CALL STATEMENT

The CALL statement permits communication between a COBOL object program and one or more COBOL subprograms or other language subprograms. Its format is:

```
CALL literal [USING identifier-1 [identifier-2]. . .]
```

Literal is a nonnumeric literal which names the program being called. The program in which the CALL statement appears is the calling program. Literal must conform to the rules for formation of a program-name. The first eight characters of literal are used to make the correspondence between the called and calling program.

When the called program is to be entered at the beginning of the Procedure Division, literal must specify the program-name in the PROGRAM-ID paragraph of the called program. The called program must have a USING clause as part of its Procedure Division header if there is a USING clause in the CALL statement that invoked it.

The identifiers specified in the USING option of the CALL statement indicate those data items available to a calling program that may be referred to in the called program.

When the called subprogram is a COBOL program, each of the operands in the USING option of the calling program must be defined as a data item in the File Section, Working-Storage Section, or Linkage Section.

Names in the two USING lists (that of the CALL in the main program and that of the Procedure Division header in the subprogram) are paired in a one-for-one correspondence.

There is no necessary relationship between the actual names used for such paired names, but the data descriptions must be equivalent. When a group data item is named in the USING list of a Procedure Division header, names subordinate to it in the subprogram's Linkage Section may be employed in subsequent subprogram procedural statements.

The USING option should be included in the CALL statement only if there is a USING option in the Procedure Division header of the called program. The number of operands in the USING option of the CALL statement should be the same as the number of operands in the USING option of the Procedure Division header. If the number of operands in the USING option of the CALL statement is greater than the number in the USING option in the called program, only those specified in the USING option of the called program may be referred to by the called program.

The execution of a CALL statement causes control to pass to the called program. The first time a called program is entered, its state is that of a fresh copy of the program. Each subsequent time a called program is entered, the state is as it was upon the last exit from that program. Thus, the reinitialization of the following items is the responsibility of the programmer:

- GO TO statements which have been altered
- TALLY
- Data items
- PERFORM statements

If a branch is made out of the range of a PERFORM, after which an exit is made from the program, the range of the PERFORM is still in effect upon a subsequent entry.

Called programs may contain CALL statements. However, a called program must not contain a CALL statement that directly or indirectly calls the calling program. A called program may be segmented.

The USING option makes the data items defined in a calling program available to a called program. The number of operands in the USING option of a called program must be less than or equal to the number of operands in the corresponding CALL statement of the invoking program or the results are unspecified.

The USING option appears in two formats which are:

Format 1 (Within a Calling Program):

```
CALL literal [USING identifier-1 [identifier-2] . . . ]
```

Format 2 (Within a Called Program):

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

The USING option must be present in the Procedure Division header if the object program is to function under the control of a CALL statement, and the CALL statement contains a USING option.

Each of the operands in the USING option of the CALL statement must have been defined as a data item in the File, Working-Storage or Linkage Section and must have a level-number of 01 or 77.

Each of the operands in the USING option of the Procedure Division header must have been defined as a data item in the Linkage Section of the program in which this header occurs, and must have a level number of 01 or 77. The compiler aligns each level-01 item on a word boundary; however, it is the programmer's responsibility to ensure proper alignment of 77 levels.

When the USING option is present, the object program operates as though each identifier in the Procedure Division had been replaced by the corresponding identifier from the USING option in the CALL statement of the calling program. That is, corresponding identifiers refer to a single set of data which is available to the calling program. The correspondence is positional and not by name.

The following is an example of a calling program with the USING option:

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CALLPROG.
           .
           .
           .
       DATA DIVISION.
           .
           .
           .
       WORKING-STORAGE SECTION.
       Ø1  RECORD-1.
           Ø3  SALARY          PICTURE  S9(5)V99.
           Ø3  RATE            PICTURE  S9V99.
           Ø3  HOURS           PICTURE  S99V9.
           .
           .
       PROCEDURE DIVISION.
           .
           .
           CALL "SUBPROG" USING RECORD-1.
```

The following is an example of a called subprogram associated with the preceding calling program:

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. SUBPROG.
           .
           .
           .
       DATA DIVISION.
           .
           .
           .
       LINKAGE SECTION.
       Ø1  PAYREC.
           Ø2  PAY                    PICTURE  S9(5)V99.
           Ø2  HOURLY-RATE            PICTURE  S9V99.
           Ø2  HOURS                  PICTURE  S99V9.
           .
           .
       PROCEDURE DIVISION USING PAYREC.
           .
           .
           .
       END-PROGRAM.
```

Processing begins in CALLPROG, which is the calling program. When the statement: CALL "SUBPROG" USING RECORD-1. is executed, control is transferred to the first statement of the Procedure Division in SUBPROG, which is the called program. In the calling program, the operand of the USING option is identified as RECORD-1.

When SUBPROG receives control, the values within RECORD-1 are made available to SUBPROG; in SUBPROG, however, they are referred to as PAYREC. Note that the PICTURE clauses for the subfields of PAYREC (described in the Linkage Section of SUBPROG), are the same as those for RECORD-1.

When processing within SUBPROG reaches the EXIT PROGRAM statement, control is returned to CALLPROG at the statement immediately following the original CALL statement. Processing then continues in CALLPROG.

In any given execution of these two programs, if the values within RECORD-1 are changed between the time of the first CALL and the second, the values passed at the time of the second CALL statement will be the changed, not the original, values. If the programmer wishes to use the original values, then he must ensure that they have been saved.

PROGRAM TERMINATION CONSIDERATIONS

The two ways to terminate a program in COBOL source language are:

- EXIT PROGRAM
- STOP RUN

Figure 8-12 shows the effects of each program termination statement based on whether it is issued within a main program or a subprogram.

Termination Statement	Main Program	Subprogram
EXIT PROGRAM	Return to system and cause end of job step.	Return to invoking program
STOP RUN	Return to system and cause end of job step.	Return to system and cause end of job step.

Figure 8-12. Effect of Program Termination Statements Within Main Programs and Subprograms

A main program is the highest level COBOL program invoked by another COBOL program. (Programs written in the other languages that follow COBOL linkage conventions are considered COBOL programs in this sense.)

If program segmentation is used, the programmer must divide the entire Procedure Division into named sections. (See *Segmentation* in Section 9.)

Execution begins with the first statement of the Procedure Division. 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.

EXIT PROGRAM STATEMENT

This form of the EXIT statement marks the logical end of a called program.

```
EXIT PROGRAM.
```

The EXIT statement must be preceded by a paragraph-name and be the only statement in the paragraph.

If control reaches an EXIT PROGRAM statement while operating under the control of a CALL statement, control returns to the point in the calling program immediately following the CALL statement.

If control reaches an EXIT PROGRAM statement and no CALL statement is active, control returns to the system which initiates an end of job step (same as STOP RUN).

STOP RUN STATEMENT

The STOP RUN statement causes execution of the object program to be terminated and control transferred to the system.

COMPILER-DIRECTING STATEMENTS

Compiler-directing statements are special statements that provide instructions for the COBOL compiler. The compiler-directing statements are ENTER and NOTE.

ENTER STATEMENT

The ENTER statement provides a means of allowing the use of more than one language in the same program. The format is as follows:

```
ENTER language-name [routine-name].
```

The ENTER statement serves only as documentation, as this compiler does not allow another source language in the program.

NOTE STATEMENT

The NOTE sentence allows the programmer to write commentary which is produced on the listing but not compiled. The format is as follows:

<u>NOTE</u> character-string.

Any combination of the characters from the computer's character set may be included in the character-string.

If a NOTE sentence is the first sentence of a paragraph, the entire paragraph is considered to be part of the character-string. Proper format rules for paragraph structure must be observed.

If a NOTE sentence appears as other than the first sentence of a paragraph, the commentary ends with the first instance of a period followed by a space.

Explanatory comments may be inserted on any line within a source program by placing an asterisk or a stroke in column 7 of the line. Any combination of the characters from the EBCDIC set may be included in Area A and Area B of that line. (See the **MRX/OS COBOL User Guide** for use of the COBOL coding form.) The asterisk or stroke and the characters will be produced on the listing, but serve no other purpose.

9. SPECIAL FEATURES

MRX COBOL provides three special features which are table handling, segmentation and the source program library facility.

TABLE HANDLING

Tables of data are common components of business data processing problems. Although the items that make up a table could be described as contiguous data items, there are two reasons why this approach is not satisfactory. First, from a documentation standpoint, the underlying homogeniety of the items would not be readily apparent; and second, the problem of making available an individual element of such a table would be severe when there is a decision as to which element is to be made available at object time.

Tables composed of contiguous data items are defined in COBOL by including the OCCURS clause in their data description entries. This clause specifies that the tiem is to be repeated as many times as stated. The item is considered to be a table-element and its name and description apply to each repetition or occurrence. Since each occurrence of a table-element does not have assigned to it a unique data-name, reference to a desired occurrence may be made only by specifying the data-name of the table element together with the occurrence number of the desired table element. The occurrence number is known as a subscript, and this technique of specifying individual table elements is called subscripting.

In order to facilitate such operations as table searching and manipulating specific items, a technique called Indexing is also available. Both subscripting and indexing are discussed below.

The number of occurrences of a table-element may be specified to be fixed or variable. If the occurrence number is given in the source program as fixed, the actual data that is entered into the table at object time may still comprise a variable number of occurrences of the table elements. Thus, not every table element need contain valid data.

TABLE DEFINITION

To define a one-dimensional table, the programmer uses an OCCURS clause as part of the data description of the table-element, but the OCCURS clause must not appear in the description of group items which contain the table-element. Example 1 shows a one-dimensional table defined by the item TABLE-ELEMENT.

Example 1:

SEQUENCE		CONT.	A	B COBOL STATEMENT
(PAGE)	(SERIAL)			
1 2 3	4 5 6	7	8 9 10 11	12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
			Ø2	TABLE-1.
			Ø3	TABLE-ELEMENT OCCURS 2Ø TIMES.
				Ø4 DOG OCCURS 5 TIMES.
				Ø5 EASY .....
				Ø5 FOX .....

In Example 2, TABLE-ELEMENT defines a one-dimensional table, but DOG does not since there is an OCCURS clause in the description of the group item (TABLE-ELEMENT) which contains DOG.

Example 2:

SEQUENCE		CONT.	A	B COBOL STATEMENT
(PAGE)	(SERIAL)			
1 2 3	4 5 6	7	8 9 10 11	12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
			Ø1	TABLE-1.
			Ø2	TABLE-ELEMENT OCCURS 2Ø TIMES.
				Ø3 DOG .....
				Ø3 FOX .....

In both examples, the complete set of occurrences of TABLE-ELEMENT has been assigned the name TABLE-1. However, it is not necessary to give a group name to the table, unless it is desired to refer to the complete table as a group item.

None of the three one-dimensional tables which appear in the following two examples have a group name.

Example 3:

SEQUENCE		CONT.	A	B COBOL STATEMENT
(PAGE)	(SERIAL)			
1 2 3	4 5 6	7	8 9 10 11	12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
			Ø1	ABLE.
				Ø2 BAKER .....
				Ø2 CHARLIE OCCURS 2Ø TIMES .....
				Ø2 DOG .....

Example 4:

SEQUENCE (PAGE)	SEQUENCE (SERIAL)	CONT.	A	B COBOL STATEMENT
1 2 3	4 5 6	7	8 9 10 11	12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
			Ø1	ABLE.
				Ø2 BAKER OCCURS 2Ø TIMES .....
				Ø2 CHARLIE .....
				Ø2 DOG OCCURS 5 TIMES .....

Defining a one-dimensional table within each occurrence of an element of another one-dimensional table gives rise to a two-dimensional table. To define a two-dimensional table, then, an OCCURS clause must appear in the data description of the element of the table, and in the description of only one group item which contains that element. Thus, in Example 5, DOG is an element of a two-dimensional table; it occurs 5 times within each element of the item BAKER which itself occurs 20 times. BAKER is an element of a one-dimensional table.

Example 5:

SEQUENCE (PAGE)	SEQUENCE (SERIAL)	CONT.	A	B COBOL STATEMENT
1 2 3	4 5 6	7	8 9 10 11	12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50
			Ø2	BAKER OCCURS 2Ø TIMES .....
				Ø3 CHARLIE .....
				Ø3 DOG OCCURS 5 TIMES .....

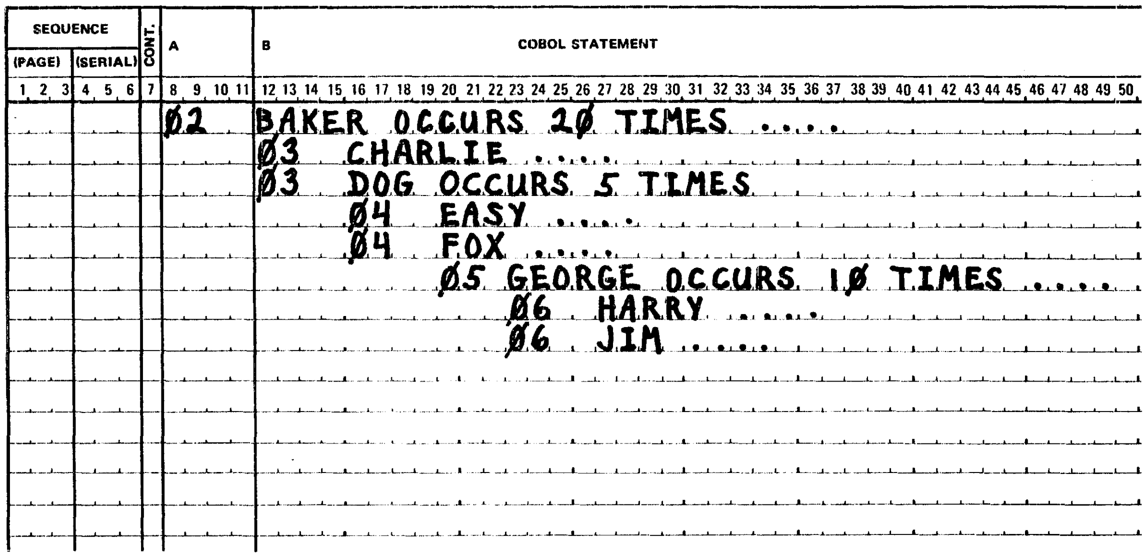
REFERENCES TO TABLE-ITEMS

Whenever the user refers to a table-element, or to an item within a table-element, the reference must indicate which occurrence of the element is intended. For access to a one-dimensional table the occurrence number of the desired element provides complete information. For tables of more than one dimension, an occurrence number must be supplied for each dimension of the table. In Example 5 then, a reference to the 4th BAKER or the 4th CHARLIE would be complete, whereas a reference to the 4th DOG would not. To refer to DOG, which is an element of a two-dimensional table, the user must refer to, for example, the 4th DOG in the 5th BAKER.

SUBSCRIPTING

One method by which occurrence numbers may be specified is to append one or more subscripts to the data-name. A subscript is an integer whose value specifies the occurrence number of an element within the group item that has the next lower level-number. The subscript can be represented either by a positive integer numeric literal, by a data-name which is defined as a numeric elementary integer, or by the special register TALLY. In any case, the subscript, enclosed in parentheses, is written immediately following the name of the table element. A table element must include as many subscripts as there are dimensions in the table whose element is being referred to. That is, there must be a subscript for each OCCURS clause in the hierarchy containing the data-name, including the data-name itself.

Example 6:



SEQUENCE (PAGE) (SERIAL) CONT. A B COBOL STATEMENT

```
       Ø2  BAKER OCCURS 1Ø TIMES .....
           Ø3  CHARLIE .....
           Ø3  DOG OCCURS 5 TIMES
               Ø4  EASY .....
               Ø4  FOX .....
                   Ø5  GEORGE OCCURS 1Ø TIMES .....
                       Ø6  HARRY .....
                       Ø6  JIM .....
```

In example 6, references to BAKER and CHARLIE require only one subscript, references to DOG, EASY, and FOX require two, and references to GEORGE, HARRY, and JIM require three.

data-name (subscript[,Δsubscript] [,Δsubscript])

The subscript, or set of subscripts, that identifies the table element is enclosed in parentheses immediately following the space that terminates data-name, which is the name of the table element. When more than one subscript appears within a pair of parentheses, the subscripts must be separated by commas. A space must follow each comma, but no space may appear between the left parenthesis and the left-most subscript or between the right-most subscript and the right parenthesis.

Restrictions on the use of a data-name as a subscript are:

1. Data-name must be a numeric elementary item that represents a positive integer.

2. The name itself may not be subscripted.

3. Data-name cannot be an index data item (item with USAGE IS INDEX).

When more than one subscript is required, they are written (separated by a comma and a space) in order corresponding to the occurrence numbers in successively less inclusive dimensions of the data organization. If a multi-dimensional table is thought of as a series of nested tables and the most inclusive or outermost table in the nest is considered to be the major table with the innermost or least inclusive table being the minor table, then the subscripts are written from left to right in the order major, intermediate, and minor. Thus, in Example 6, a reference to HARRY (18, 2, 7) means the HARRY in the 7th GEORGE, in the 2nd DOG, in the 18th BAKER.

A reference to an item must not be subscripted if the item is not a table-element or an item or condition-name within a table-element.

The lowest permissible subscript value is 1. The highest permissible subscript value in any particular case is the maximum number of occurrences of the item as specified in the OCCURS clause.

When a data-name is used as a subscript, it may be used to refer to items within many different tables. These tables need not have elements of the same size. The data-name may also appear as the only subscript with one item and as one of two or three subscripts with another item. Also, it is permissible to mix literal and data-name subscripts, for example, HARRY (12, NEWKEY, 2).

INDEXING

References can be made to individual elements within a table of elements by specifying indexing for that reference.

An index is assigned to a given level of a table by using an INDEXED BY clause in the definition of the table. A name given in the INDEXED BY clause is known as an index-name and is used to refer to the assigned index. An index-name must be initialized by a SET statement before it is used in a table reference. An index may be modified only by a SET statement. Data items described by the USAGE IS INDEX clause permit storage of the values of index-names as data without conversion. Such data items are called index data items.

Direct indexing is specified by using an index-name in the form of a subscript. The format is as follows:

```
data-name (index-name-1 [,Δindex-name-2] [,Δindex-name-3] )
```

Relative indexing is specified when the terminal space of the data-name is followed by a parenthesized group of items: the index-name, followed by a space, one of the operators + or -, another space, and an unsigned integral numeric literal. The format is as follows:

```
data-name (index-name-1 [ {±} integer] [,Δindex-name[ {±} integer] ]

          [,Δindex-name-3[ {±} integer] ] )
```

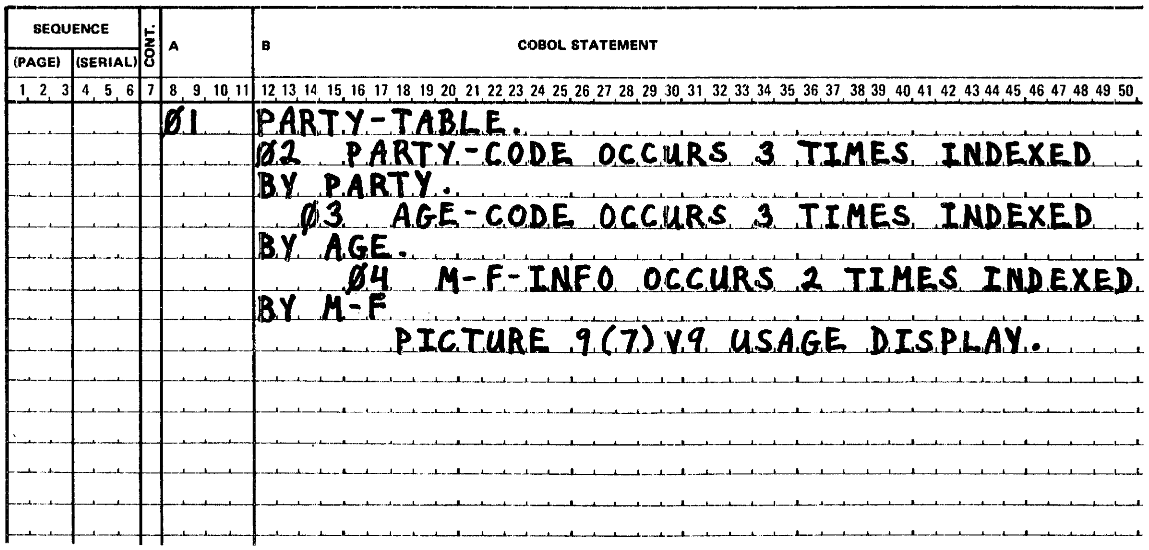
RESTRICTIONS ON INDEXING AND SUBSCRIPTING

Tables may have one, two, or three dimensions. Therefore, references to an element in a table may require up to three subscripts or indexes.

1. A data-name must not be subscripted or indexed when the data name is itself being used as an index or subscript.
2. Subscripting and indexing must not be used together in a single reference.
3. Wherever subscripting is not permitted, indexing is not permitted.
4. The commas shown in the formats for indexes and subscripts are required.
5. The syntax rules for indexing are the same as those for subscripting.

EXAMPLES OF SUBSCRIPTING AND INDEXING

For a table with three levels of indexing, the following Data Division entries would result in a storage layout as shown in Figure 9-1.



```
SEQUENCE        CONT. A    B                 COBOL STATEMENT
(PAGE) (SERIAL)
1 2 3  4 5 6    7     8 9 10 11 12 ... 50
                      Ø1   PARTY-TABLE.
                           Ø2  PARTY-CODE OCCURS 3 TIMES INDEXED
                           BY PARTY.
                              Ø3  AGE-CODE OCCURS 3 TIMES INDEXED
                           BY AGE.
                                 Ø4  M-F-INFO OCCURS 2 TIMES INDEXED
                           BY M-F
                                    PICTURE 9(7)V9 USAGE DISPLAY.
```

PARTY-TABLE contains three levels of indexing. Reference to elementary items within PARTY-TABLE is made by use of a name that is subscripted or indexed. A typical Procedure Division statement might be:

MOVE M-F-INFO (PARTY, AGE, M-F) to M-F-RECORD.

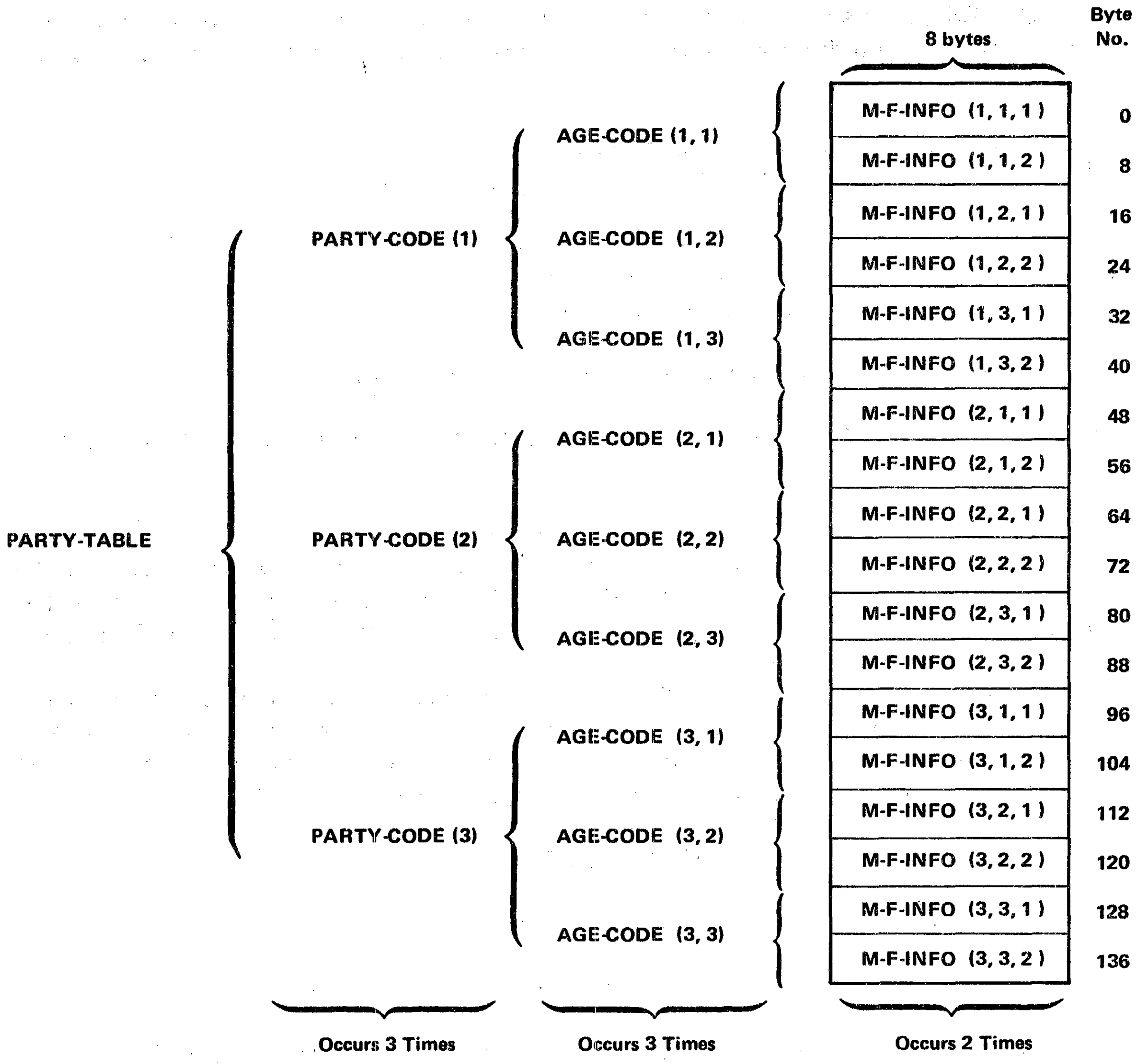


Figure 9-1. Example of Table Indexing

DATA DIVISION CONSIDERATIONS FOR TABLE HANDLING

The OCCURS and USAGE clauses are included as part of the record description entries in a program utilizing the table handling feature.

OCCURS CLAUSE

The OCCURS clause eliminates the need for separate entries for repeated data and supplies information required for the application of subscripts or indexes. The clause has two formats which are:

```
Format 1:     OCCURS integer-2 TIMES
              [INDEXED BY index-name-1 [index-name-2]. . .]

Format 2:     OCCURS integer-1 TO integer-2 TIMES
              [DEPENDING ON data-name-1]
              [INDEXED BY index-name-1 [index-name-2]. . .]
```

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

In Format 1, integer-2 represents the exact number of occurrences. It must be greater than zero.

In Format 2, the DEPENDING ON option is used. This indicates that the subject of this entry has a variable number of occurrences. This does not mean that the length of the subject is variable, but rather that the number of times the subject may be repeated is variable, the number of times being controlled by the value of data-name-1 at object time.

In Format 2, integer-1 represents the minimum number of occurrences, and integer-2 represents the maximum number of occurrences. Integer-1 may be zero or any positive integer. Integer-2 must be greater than zero, and also greater than integer-1. Integer-2 must be less than 16,384. The value of data-name-1 must not exceed integer-2.

Data-name-1, the object of the DEPENDING ON option:

- Must be described as a positive integer
- Must not exceed integer-2 in value
- Must not be subscripted (that is, must not itself be the subject of, or an entry within, a table)
- Must, if it appears in the same record as the table it controls, appear before the variable portion of the record

The subject of an OCCURS clause is the data-name of the entry that contains this OCCURS clause. The subject of an OCCURS clause must be subscripted or indexed whenever referenced. When subscripted, the subject refers to one occurrence within the table.

The OCCURS clause may not be specified in a data description entry that has a level-01 or level-77 number, or an entry that describes an item whose size is variable.

The DEPENDING option is only required when the end of the occurrences cannot otherwise be determined. Unused character positions resulting from the DEPENDING ON option will appear on the external media. DEPENDING ON may appear only once in a record description entry and must be at the last major level of the record.

The total number of index-names for a program must not exceed 255.

An INDEXED BY phrase is required if the subject of this entry (or an item within it) is to be referred to by indexing. The index-name identified in this clause is not defined elsewhere since its allocation and format are dependent on the system. Not being data, it cannot be associated with any data hierarchy.

There are two types of indexing: direct indexing and relative indexing.

Direct indexing: If a data-name is used in the procedure text with index-names, the data-name itself must be the subject of an INDEXED BY option, or be subordinate to a group(s) that is the subject of the INDEXED BY option.

The following example:

    A (INDEX-1, INDEX-2, INDEX-3)

implies that A belongs to a structure with three levels of OCCURS options, each with an INDEXED BY option.

Relative Indexing: The index-name is followed by a space, followed by one of the operators + or -, followed by another space, followed by an unsigned numeric literal. The numeric literal is considered to be an occurrence number, and is converted to an index value before being added to, or subtracted from, the corresponding index-name.

Given the following example:

    A (Z + 1, J + 3, K + 4)

where:

    table element indexed by Z has an entry length of 100

    table element indexed by J has an entry length of 10

    table element indexed by K has an entry length of 2

the resulting address will be computed as follows:

$$(\text{ADDRESS of A}) + Z + \underbrace{\underline{100 * 1}}_{} + J + \underbrace{\underline{10 * 3}}_{} + K + \underbrace{\underline{4 * 2}}_{}$$

Conversion of integers to
index values

An index-name must be initialized through a SET statement before it is used. Each index-name is a word in length and contains a binary value that represents an actual displacement from the beginning of the table that corresponds to an occurrence number in the table. The value is calculated as the occurrence number minus one, multiplied by the length of the entry that is indexed by this index-name.

For example, if the programmer writes:

```
A OCCURS 15 TIMES INDEXED BY Z PICTURE IS X(10)
```

on the fifth occurrence of A, the binary value contained in Z will be:

```
Z = (5 - 1) * 10 = 40
```

Any entry which contains or has a subordinate entry which contains Format 2 cannot be the object of the REDEFINES clause.

The VALUE clause must not be stated in a data description entry which contains an OCCURS clause or in an entry which is subordinate to an entry containing an OCCURS clause.

USAGE CLAUSE

The USAGE clause specifies the format of a data item in the computer storage. The format is as follows:

[<u>USAGE</u> IS] <u>INDEX</u>

The USAGE clause can be written at any level. If the USAGE clause is written at a group level, it applies to each elementary item in the group. The USAGE clause of an elementary item cannot contradict the USAGE clause of a group to which the item belongs.

An elementary item described with the USAGE IS INDEX clause is called an index data item and can be used to save index-name values for future reference. An index data item must be assigned an index-name value (that is (occurrence number - 1) * entry length) through the SET statement. Such a value corresponds to an occurrence number in a table.

If a group 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 SET statements or in a relation condition.

An index data item can be referred to directly only in a SET statement or in a relation condition. An index data item can be part of a group which is referred to in a MOVE or input-output statement, in which case no conversion will take place.

The SYNCHRONIZED, JUSTIFIED, PICTURE, VALUE and BLANK WHEN ZERO clauses cannot be used to describe group or elementary items described with the USAGE IS INDEX clause.

PROCEDURE DIVISION CONSIDERATIONS FOR TABLE HANDLING

The SET statement may be used to facilitate table handling. In addition, there are special rules involving table handling elements when they are used in relation conditions.

RELATION CONDITION

The result of the comparison of two index-names and/or index data items 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.

The result of the comparison of an index data item with any data item not specified above is illegal and the result is unpredictable.

Figure 9-2 shows permissible comparisons for index-names and index data items.

Second Operand / First Operand	Index-Name	Index Data Item	Data-Name (numeric integer only)	Numeric Literal (integer only)
Index-Name	Compare occurrence number	Compare without conversion	Compare occurrence number with data-name	Compare occurrence number with literal
Index-Data Item	Compare without conversion	Compare without conversion	Illegal	Illegal
Data-Name (numeric integer only)	Compare occurrence number with data-name	Illegal	See Table 8-1 for permissible comparisons	
Numeric Literal (integer only)	Compare occurrence number with literal	Illegal		

Figure 9-2. Index-Names and Index Data Items — Permissible Comparisons

SET STATEMENT

The SET statement establishes reference points for table-handling operations by setting index-names associated with table elements. It has two formats which are:

```
Format 1:      SET {index-name-1  [index-name-2]...} TO
                   {identifier-1  [identifier-2] ...}

                   {index-name-3}
                   {identifier-3}
                   {literal-1   }

Format 2:      SET  index-name-4  [index-name-5]...

                   {UP BY  }  {identifier-4}
                   {DOWN BY}  {literal-2   }
```

All references to index-name-1, identifier-1, and index-name-4 apply equally to index-name-2, identifier-2, and index-name-5, respectively.

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 a literal is used, it must be a positive integer. Index-names are considered related to a given table and are defined by being specified in the INDEXED BY clause.

There may not be more than 20 operands in a SET statement.

The maximum value of literal-1 or literal-2 is 2^{16}-1. Thus a literal with more than 4 digits may lose significance.

In Format 1, the following action occurs:

- Index-name-1 is set to a value that corresponds to the same occurrence number to which either index-name-3, identifier-3, or literal-1 corresponds. If identifier-3 is an index data item, or if index-name-3 is related to the same table as index-name-1, no conversion takes place.

- 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. Literal-1 cannot be used in this case.

- 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 literal-1 can be used in this case.

- The process is repeated for subsequent index-names or identifiers, if specified. Each time the value of index-name-3 or identifier-3 is used, it is used as it was at the beginning of the execution of the statement. Any subscripting or indexing associated with an identifier is evaluated immediately before the value of the respective data item is changed.

In Format 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 literal-2 or identifier-4. This process is repeated for subsequent index-names. Each time the value of identifier-4 is used, it is used as it was at the beginning of the execution of the statement.

SEGMENTATION

COBOL segmentation is a facility that provides a means of specifying object program overlay requirements to the compiler.

COBOL segmentation deals only with segmentation of procedures. As such, only the Procedure Division and the Environment Division are considered in determining segmentation requirements for an object program.

ORGANIZATION

Although it is not mandatory, the Procedure Division for a source program is usually written as a consecutive group of sections, each of which is composed of a series of closely related operations that are designed to collectively perform a particular function. However, when segmentation is used, the entire Procedure Division must be in sections. In addition, each section must be classified as belonging either to the fixed portion or to one of the independent segments of the object program.

FIXED PORTION

The fixed portion is defined as that part of the object program which is logically treated as if it were always in memory. This portion of the program is composed of two types of segments: permanent segments and overlayable fixed segments.

A permanent segment is a segment in the fixed portion which cannot be overlayed by any other part of the program. An overlayable fixed segment is a segment in the fixed portion which, although logically treated as if it were always in memory, can be overlayed, if necessary, by another segment to optimize memory utilization. However, such a segment, if called for by the program, is always available in its last used state.

Also, depending on the availability of memory, the number of permanent segments in the fixed portion can be varied using a special facility called SEGMENT-LIMIT.

INDEPENDENT SEGMENTS

An independent segment is defined as part of the object program which can overlay, and can be overlayed by, either an overlayable fixed segment or another independent segment. An independent segment is effectively in its initial state each time the segment is made available to the program.

SEGMENT CLASSIFICATION

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

- Logic Requirements — Sections which must be available for reference at all times, or which are referred to very frequently, are normally classified as belonging to one of the permanent segments; sections which are used less frequently are normally classified as belonging either to one of the overlayable fixed segments or to one of the independent segments, depending on logic requirements.
- Frequency of Use — Generally, the more frequently a section is referred to, the lower its priority-number; the less frequently it is referred to, the higher its priority-number.
- Relationship to Other Sections — Sections which frequently communicate with one another should be given the same priority-numbers.

SEGMENTATION CONTROL

The logical sequence of the program is the same as the physical sequence except for specific transfers of control. The compiler will provide transfers to maintain the logic flow of the source program. The compiler will also insert instructions necessary to load and/or initialize a segment when necessary. Control may be transferred within a source program to any paragraph in a section; that is, it is not mandatory to transfer control to the beginning of a section.

Sections of a given segment may be scattered throughout the source program.

STRUCTURE OF PROGRAM SEGMENTS

Program segments are made up of sections classified according to priority numbers.

PRIORITY NUMBERS

Section classification is accomplished by means of a system of priority-numbers. The priority-number is included in the section header. The format is as follows:

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

The priority-number must be an integer ranging in value from 0 through 99. If the priority-number is omitted from the section header, the priority is assumed to be 0.

All sections which have the same priority-number constitute a program segment with that priority. Segments with priority-number 0 through 49 belong to the fixed portion of the object program. Segments with priority-number 50 through 99 are independent segments.

SEGMENT-LIMIT

Ideally, all program segments having priority-numbers ranging from 0 through 49 should be specified as permanent segments. However, when insufficient memory is available to contain all permanent segments plus the largest overlayable segment, it becomes necessary to decrease the number of permanent segments. The SEGMENT-LIMIT feature provides the user with a means by which he can reduce the number of permanent segments in his program, while still retaining the logical properties of fixed portion segments (priority-numbers 0 through 49).

The SEGMENT-LIMIT clause appears in the OBJECT-COMPUTER paragraph (Environment Division) and has the following format:

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

Priority-number must be an integer ranging in value from 1 through 49.

When the SEGMENT-LIMIT clause is specified, only those segments having priority-numbers from 0 up to, but not including, the priority number designated as the segment-limit, are considered as permanent segments of the object program. Those segments having priority-numbers from the segment-limit through 49 are considered as overlayable fixed segments.

When the SEGMENT-LIMIT clause is omitted, all segments having priority-numbers from 0 through 49 are considered as permanent segments of the object program.

RESTRICTIONS ON PROGRAM FLOW

When segmentation is used, the following restrictions are placed on the ALTER and the PERFORM statements.

ALTER STATEMENT

A GO TO statement in a section whose priority is equal to or higher than 50 must not be referred to by an ALTER statement in a section with a different priority.

All other uses of the ALTER statement are valid and are performed even if the GO TO to which the ALTER refers is in a segment of the program that has not yet been called for execution.

PERFORM STATEMENT

A PERFORM statement that appears in a section whose priority-number is less than the segment-limit, can have within its range only the following:

- Sections each of which has a priority-number less than 50.

- Sections wholly contained in a single segment whose priority-number is greater than 49.

A PERFORM statement that appears in a section whose priority-number is equal to or greater than the segment-limit, can have within its range only the following:

- Sections each of which has the same priority-number as that containing the PERFORM statement.

- Sections with a priority-number that is less than the segment-limit.

When a procedure-name in a segment with a priority-number greater than 49 is referred to by a PERFORM statement contained in a segment with a different priority-number, the segment referred to is made available in its initial state for each execution of the PERFORM statement.

SOURCE PROGRAM LIBRARY FACILITY

Prewritten source program entries can be included in a source program. Thus, an installation can use standard file descriptions, record descriptions, or procedures, without recording them. These entries and procedures are contained in user-created libraries; they are included in a source program by means of a COPY statement.

The system UPDATE function must be used to perform the COPY functions. (Refer to **MRX/OS Program Library Services Reference** manual for a description of the Librarian UPDATE.)

A. GLOSSARY OF COBOL TERMS

The terms in this appendix are defined in accordance with their meaning as used in this document describing COBOL and may not have the same meaning for other languages.

Notes affixed to some of the definitions have the following meanings:

(1) The definition agrees with the IFIP-ICC Vocabulary of Information Processing.*

(2) The definition is more restrictive than the corresponding definition in the IFIP-ICC Vocabulary of Information Processing.

(3) The definition is more inclusive than the corresponding definition in the IFIP-ICC Vocabulary of Information Processing.

(4) The definition disagrees with the IFIP-ICC Vocabulary of Information Processing.

Access, Random — An access mode in which specific logical records are obtained from or placed in a mass storage file in a non-sequential manner.

Access, Sequential — An access mode in which a logical record read from or written to a file has an implicit logical predecessor and an implicit logical successor. The first access to a file accesses a record that has no implicit logical predecessor; each successive access refers to the implicit logical successor of the previously accessed logical record. The predecessor/successor relationships of a record are established when the record is written to a file.

Actual Decimal Point — (See Decimal Point, Actual)

Actual Key — (See Key, Actual)

Alphabetic Character — (See Character, Alphabetic)

Alphanumeric Character — (See Character, Alphanumeric)

Assumed Decimal Point — (See Decimal Point, Assumed)

*International Federation for Information Processing — International Computation Center Vocabulary for Information Processing, 1st English Language Edition, 1966, North-Holland Publishing Company, Amsterdam.

Block (3) – A physical unit of data that is convenient to a particular computer for storage on an input or output device. The term is synonymous with physical record. The block is normally composed of one or more logical records. The size of a block has no direct relationship to the size of the file within which the block is contained or to the size of the logical record(s) that are contained within the block.

Character (1) – The basic indivisible unit of the language.

Character, Alphabetic (2) – A character that belongs to the following set of letters:

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

Character, Alphanumeric (1) – Any character in the computer's character set.

Character, Editing – A single character or a fixed two-character combination belonging to the following set:

Character	Meaning
B	Space
0	Zero
+	Plus
-	Minus
CR	Credit
DB	Debit
Z	Zero suppress
*	Check protect
$	Currency sign
,	Comma (decimal point)
.	Period (decimal point)

Character, Numeric (1) – A character that belongs to the following set of digits:

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

Character, Punctuation – A character that belongs to the following set:

Character	Meaning
,	Comma
.	Period
" or '	Quotation mark
(	Left parenthesis
)	Right parenthesis
	Space

Character, Special (1) — A character that belongs to the following set:

Character	Meaning
+	Plus sign
-	Minus sign
*	Asterisk
/	Stroke (virgule, slash)
$	Currency sign
,	Comma (decimal point)
.	Period (decimal point)
" or '	Quotation mark
(	Left parenthesis
)	Right parenthesis

Characters, Standard — A character-string that comprises a data item whose size is measured in accordance with standard data format.

Character Set (1) — The complete COBOL character set consists of the characters listed below:

Character	Meaning
0,1,...,9	Digit
A,B,...,Z	Letter
	Space (blank)
+	Plus sign
-	Minus sign (hyphen)
*	Asterisk
/	Stroke (virgule, slash)
$	Currency sign
,	Comma (decimal point)
.	Period (decimal point)
" or '	Quotation mark
(	Left parenthesis
)	Right parenthesis

Character-String — Contiguous characters which form a literal, a word, a PICTURE in the Data Division, or a NOTE in the Procedure Division. The rules governing the construction of each of the above types of character-strings differ, and are explained in other chapters.

Class Condition — (See Condition, Class)

Clause — A clause is an ordered set of consecutive COBOL words whose purpose is to specify an attribute of an entry.

Clause, Data — A clause that appears in a data description entry in the Data Division and provides information describing a particular attribute of a data item.

Clause, Environment — A clause that appears as part of an Environment Division entry.

Clause, File — A clause that appears as part of a file description (FD) in the Data Division.

COBOL Object Program — (See Object Program, COBOL)

COBOL Source Program — (See Source Program, COBOL)

Collating Sequence — (See Sequence, Collating)

Comment — An annotation in the Identification Division or Procedure Division of a source program.

Compile Time — (See Time, Compile)

Compiler Directing Statement — (See Statement, Compiler Directing)

Condition — A simple condition, or a syntactically correct combination of simple conditions and logical operators, for which a truth value can be determined.

Condition, Class — The proposition, for which a truth value can be determined, that the content of an item is wholly alphabetic or is wholly numeric.

Condition, Invalid Key — A condition in which, at object time, a specific value of the actual key associated with a mass storage file is determined to lie outside the limits of the file being accessed.

Condition, Relation — The proposition, for which a truth value can be determined, that the value of a data item has a specific relationship to the value of another data item. (See Operator Relational.)

Condition, Simple — Any single condition chosen from the set:

Relation condition
Class condition

Conditional Statement — (See Statement, Conditional)

Conditional Variable — (See Variable, Conditional)

CONFIGURATION SECTION — (See Section, Configuration)

Connective — A word or a punctuation character that is used to link two or more operands written in a series. (The comma is the only connective allowed, and may only appear within a subscript.)

Constant, Figurative (1) — A reserved word that represents a numeric value, a character, or a string of characters.

Constant, Literal — (See Literal)

Contiguous Items – (See Items, Contiguous)

Counter (3) – A data item used for storing numbers or number representations in a manner that permits these numbers to be increased or decreased by the value of another number, or to be changed or reset to zero or to an arbitrary positive or negative value.

Data Clause – (See Clause, Data)

Data Description Entry – (See Entry, Data Description)

Data Item (1) – Any elementary item, a named group of elementary items within a record, or a record.

Data Item, Index (4) – A data item in which the value associated with an index-name can be stored as data without conversion.

Data-Name (1) – A word that contains at least one alphabetic character and that names an entry in the Data Division. When used in the General Formats, 'data-name' represents a word which can neither be subscripted nor indexed, unless specifically permitted by the rules for that format.

Data-Name, Indexed (1) – An identifier that is composed of a data-name, followed by one or more index-names enclosed in parentheses.

Data-Name, Subscripted (1) – An identifier that is composed of a data-name followed by one or more subscripts enclosed in parentheses.

Decimal Point, Actual (2) – The physical representation of the decimal point position in a data item. Either of the decimal point characters, period (.) or comma (,) may be used for this representation.

Decimal Point Assumed – A decimal point position which does not involve the existence of an actual character in a data item. The assumed decimal point has logical meaning but no physical representation.

Division – One or more sections or paragraphs that are formed and combined in accordance with a specific set of rules. There are four (4) divisions in a COBOL program:

IDENTIFICATION
ENVIRONMENT
DATA
PROCEDURE

Division Header – (See Header, Division)

Editing Character – (See Character, Editing)

Element, Table – (See Item, Elementary)

End of Procedure Division – The physical position in a COBOL source program after which no further procedures appear.

Entry (4) – Any descriptive set of consecutive clauses terminated by a period and written in the Identification Division, Environment Division, or Data Division of a COBOL source program.

Entry, Data Description – An entry in the Data Division that is composed of a level number followed by a data-name (if required) and a set of data clauses (as required).

Entry, File Description – An entry in the File Section of the Data Division that is composed of the level indicator FD, followed by a file-name, and then followed by a set of file clauses as required.

Entry, Object of – A set of operands and reserved words, within a Data Division entry, that immediately follows the subject of the entry.

Entry, Subject of – An operand or reserved word that appears immediately following the level indicator or the level number in a Data Division entry.

Environment Clause – (See Clause, Environment)

Execution Time – (See Time, Object)

Figurative Constant – (See Constant, Figurative)

File – A collection of records.

File Clause – (See Clause, File)

File Description Entry – (See Entry, File Description)

File Limit – A set of logical boundary locations for a particular mass storage file that are within the physical boundary locations of a mass storage medium.

File, Mass Storage (2) – A collection of records that is assigned to a mass storage medium.

File-Control – The name of an Environment Division paragraph in which the data files for a given source program are declared.

File-Name – A word with at least one alphabetic character that names a file described in the Data Division.

Format (1) – A specific arrangement of a set of data.

Format, Reference – A format provides a standard method for describing COBOL source programs.

Format, Standard Data — The concept used in describing the characteristics of data in a COBOL Data Division. The characteristics or properties of the data are expressed in a form oriented to the appearance of the data on a printed page of infinite length and breadth, rather than a form oriented to the manner in which the data is stored internally in the computer, or on a particular external medium.

Header, Division — COBOL words that indicate the beginning of a particular division. The division headers are:

IDENTIFICATION DIVISION
ENVIRONMENT DIVISION
DATA DIVISION
PROCEDURE DIVISION

Header, Paragraph — A reserved word, immediately followed by a period, that precedes and identifies all entries in the Identification and Environment Division. The paragraph headers are:

In the Identification Division:

PROGRAM-ID.
AUTHOR.
INSTALLATION.
DATE-WRITTEN.
SECURITY.
REMARKS.

In the Environment Division:

SOURCE-COMPUTER.
OBJECT-COMPUTER.
SPECIAL-NAMES.
FILE-CONTROL.
I-O CONTROL.

Header, Section — A combination of words that precedes and identifies each section in the Environment, Data and Procedure Divisions.

In the Environment and Data Divisions, the permissible section headers are composed of reserved words.

In the Environment Division:

CONFIGURATION SECTION.
INPUT-OUTPUT SECTION.

In the Data Division:

FILE SECTION.
WORKING-STORAGE SECTION.
LINKAGE SECTION.

In the Procedure Division, the section header is composed of a section-name followed by the reserved word SECTION, an optional priority number and a period.

High Order End — The leftmost character of a string of characters.

Identifier (3) — The data-name, followed, as required, by the syntactically correct combination of subscripts and indexes necessary to make unique reference to a data item.

Imperative Statement — (See Statement, Imperative)

Implementor-Name — A reserved word that refers to a particular feature available on a MEMOREX computer system.

Index (4) — A computer storage position or register, the contents of which represent the identification of a particular element in a table.

Index-Name — A word with at least one alphabetic character that names an index associated with a specific table.

Index Data Item — (See Data Item, Index)

INPUT-OUTPUT SECTION — (See Section, Input-Output)

Integer — A numeric literal or a numeric data item that does not include any character positions to the right of the assumed decimal point. Where the term 'integer' appears in general formats, integer must be a numeric data item, and must be unsigned.

Invalid Key Condition — (See Condition, Invalid Key)

I-O CONTROL — The name of an Environment Division paragraph in which object program requirements for rerun points, sharing of same areas by several data files, and multiple file storage on a single input-output device are specified.

Items, Contiguous — Data items that are described by consecutive entries in the Data Division, and that bear a definite hierarchic relationship to each other.

Item, Elementary — A data item that is described as not being further logically subdivided.

Item, Group — A named contiguous set of elementary or group items.

Item, Noncontiguous — A data item, in the Working-Storage or Linkage Section, that bears no hierarchic relationship to other items.

Item Nonnumeric — A data item whose description permits its contents to be composed of any combination of characters taken from the computer's character set. Certain categories of nonnumeric items may be formed from more restricted character sets.

Item, Numeric — A data item whose description restricts its contents to a value represented by characters chosen from the digits 0 through 9, with or without an operational sign.

Key (1) – One or more data items, the contents of which jointly serve to identify the location of a record or the ordering of data.

Key, Actual (2) – A key that directly expresses the physical location of a logical record on a mass storage medium.

Key, Forward – A key used with an indexed file when it is accessed sequentially.

Key Word – (See Word, Key)

Level Indicator – Two alphabetic characters that identify a specific type of file or a position in a hierarchy.

Level-Number – Two characters that in the case of the numbers 1 to 49, indicate the hierarchical structure of a logical record, or, in the case of the number 77 identify special properties of a data description entry.

Literal (1) – A string of characters whose value is implied by the ordered set of characters comprising the string.

Literal, Nonnumeric (2) – A string of characters bounded by quotation marks. The string of characters may include any character in the computer's character set, with the exception of the quotation mark.

Literal, Numeric (2) – A literal composed of one or more numeric characters not bounded by quotation marks. It may contain either a decimal point, that cannot be the rightmost character, or an algebraic sign, that must be the leftmost character, or both.

Literal Constant – (See Literal)

Logical Record – (See Record, Logical)

Low Order End – The rightmost character of a string of characters.

Mass Storage – A storage medium on which data may be organized and maintained in both a sequential and nonsequential manner.

Mass Storage File – (See File, Mass Storage)

Mass Storage File Segment – A part of a mass storage file whose beginning and end is defined by the FILE-LIMITS clause in the Environment Division.

Mnemonic-Name – A word, supplied by the programmer, that is associated in the Environment Division with a specific implementor-name.

Noncontiguous Item – (See Item, Noncontiguous)

Nonnumeric Item – (See Item, Nonnumeric)

Nonnumeric Literal – (See Literal, Nonnumeric)

Numeric Character – (See Character, Numeric)

Numeric Item – (See Item, Numeric)

Numeric Literal – (See Literal, Numeric)

OBJECT-COMPUTER (2) – The name of an Environment Division paragraph which describes the computer environment within which the object program is executed.

Object of Entry – (See Entry, Object of)

Object Program, COBOL (2) – The set of computer instructions that are an output of the compilation of a COBOL source program.

Object Time – (See Time, Object)

Operand – Any lower case word (or words) that appear in a statement or entry format in this publication.

Operation Sign – (See Sign, Operational)

Operator, Relational – A reserved word or a group of consecutive reserved words used in the construction of a relation condition. The permissible operators and their meaning are:

Relational Operator	Meaning
IS [NOT] GREATER THAN	Greater than or not greater than
IS [NOT] LESS THAN	Less than or not less than
IS [NOT] EQUAL TO	Equal to or not equal to

Optional Word – (See Word, Optional)

Paragraph – A paragraph-name (in the Procedure Division) followed by one or more sentences, or a paragraph-header (in the Identification or Environment Divisions) followed by one or more entries.

Paragraph-Name – A word that begins and identifies a paragraph in the Procedure Division.

Paragraph Header – (See Header, Paragraph)

Physical Record – (See Block)

Priority-Number – A number, ranging in value from 0 to 99, that classifies source program sections in the Procedure Division in order to guide object program segmentation.

Procedure — A paragraph or group of logically successive paragraphs, or a section or group of logically successive sections, within the Procedure Division.

Procedure-Name — A word used to refer to a paragraph or section in the source program in which it occurs. It consists of a paragraph-name or a section-name.

Program-Name — A word that identifies a COBOL source program.

Punctuation Character — (See Character, Punctuation)

Random Access — (See Access, Random)

Record — (See Record, Logical)

Record Description — The total set of data description entries associated with a particular record.

Record, Logical (1) — The most inclusive data item.

Record, Physical — (See Block)

Record-Name (2) — A data-name that names a record.

Reference Format — (See Format, Reference)

Registers, Special — Compiler generated storage areas whose primary use is to store information produced in conjunction with the use of specific COBOL features.

Relation — (See Operator, Relational)

Relation Character — (See Character, Relation)

Relation Condition — (See Condition, Relation)

Relational Operator — (See Operator, Relational)

Reserved Word — (See Word, Reserved)

Section — A set of one or more paragraphs or entries, the first of which is preceded by a section header.

Section, Configuration — A section of the Environment Division that describes overall specifications of source and object computers.

Section, File — The section of the Data Division that contains file description entries.

Section Header — (See Header, Section)

Section, Input-Output — The section of the Environment Division that names the files and the external media required by an object program. It also provides information required for transmission and handling of data during execution of the object program.

Section, Linkage — The section of the Data Division that describes data made available from another program.

Section, Working-Storage — The section of the Data Division that describes working storage data items, composed either of noncontiguous items or of working storage records or of both.

Section-Name — A word that identifies a section written in the Procedure Division. (See Word.)

Sentence — A sequence of one or more statements, the last of which is terminated by a period followed by a space.

Sequence, Collating (1) — The MRX defined sequence in which the characters that are acceptable to a computer are ordered for purposes of comparison.

Sequential Access — (See Access, Sequential)

Sign, Operational — An algebraic sign, associated with a numeric literal, to indicate whether the item is positive or negative.

Simple Condition — (See Condition, Simple)

SOURCE COMPUTER — The name of an Environment Division paragraph which describes the computer environment within which the source program is compiled.

Source Program, COBOL — A program coded in COBOL language that must be translated into machine language before use.

Special Character — (See Character, Special)

Special Registers — (See Registers, Special)

SPECIAL-NAMES — The name of an Environment Division paragraph in which implementor-names are related to user specified mnemonic-names.

Standard Characters — (See Characters, Standard)

Standard Data Format — (See Format, Standard Data)

Statement — A syntactically valid combination of words and symbols written in the Procedure Division beginning with a verb.

Statement, Compiler Directing — A statement, beginning with a compiler directing verb, that causes the compiler to take a specific action during compilation.

Statement, Conditional — 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.

Statement, Imperative — A statement that begins with an imperative verb and specifies an unconditional action to be taken. An imperative statement may consist of a sequence of imperative statements.

Subject of Entry — (See Entry, Subject of)

Subscript (1) — An integer whose value identifies a particular element in a table.

Subscripted Data-Name — (See Data-Name, Subscripted)

System Name — A word that specifies the external name of a file, a device, and an organization method.

Table (1) — A set of logically consecutive items of data that are defined in the Data Division by means of the OCCURS clause.

Table Element — (See Item, Elementary)

Time, Compile — The time at which a COBOL source program is translated, by a COBOL compiler, to a COBOL object program.

Time, Execution — (See Time, Object)

Time, Object — The time at which an Object Program is executed.

Truth Value — The representation of the result of the evaluation of a condition in terms of one of two values:

True
False

Unit — A single lower-case word or a group of lower-case words and one or more reserved words enclosed in brackets or braces.

Variable — A data item whose value may be changed by execution of the object program.

Variable, Conditional — A data item consisting of one or more values which has a condition-name assigned to it.

Verb — A word that expresses an action to be taken by a COBOL compiler or object program.

Word (2) — A word is a sequence of not more than 30 characters. Each character is selected from the set A through Z, 0 through 9, and - except that the '-' may not appear as the first or last character in a word. A word is delimited by separators.

Word, Key (2) – A reserved word whose presence is required when the format in which the word appears is used in a source program.

Word, Optional (2) – A reserved word that is included in a specific format only to improve the readability of the language and whose presence is optional to the user when the format in which the word appears is used in a source program.

Word, Reserved (1) – One of a specified list of words which may be used in COBOL source programs, but which must not appear in the programs as user-defined words.

WORKING-STORAGE SECTION – (See Section, Working-Storage)

B. EBCDIC COLLATING SEQUENCE

The EBCDIC collating sequence in ascending order is as follows.

Character	Meaning
	Space
.	Period or decimal point
<	Less than symbol
(	Left parenthesis
+	Plus symbol
$	Currency symbol
*	Asterisk
)	Right parenthesis
;	Semicolon
-	Hyphen or minus symbol
/	Stroke, virgule, or dash
,	Comma
>	Greater than symbol
'	Apostrophe or single quotation mark
=	Equal sign
"	Quotation mark
A through Z	Alphabet
0 through 9	Numerals

C. MRX COBOL RESERVED WORDS

ACCEPT
ACCESS
ACTUAL
ADD
ADDRESS
ADVANCING
AFTER
ALL
ALPHABETIC
ALTER
ALTERNATE
AND
ARE
AREA
AREAS
ASCENDING
ASSIGN
AT
AUTHOR

BEFORE
BEGINNING
BINARY
BLANK
BLOCK
BY

CALL
CF
CH
CHARACTERS
CLOCK-UNITS
CLOSE
COBOL
CODE
COLUMN
COMMA
COMP
COMP-1
COMP-2
COMP-3
COMPUTATIONAL
COMPUTATIONAL-1
COMPUTATIONAL-2
COMPUTATIONAL-3
COMPUTE
CONFIGURATION
CONSOLE
CONTAINS
CONTROL
CONTROLS
COPY
CORR
CORRESPONDING
CURRENCY

DATA
DATE-COMPILED
DATE-WRITTEN
DE
DECIMAL-POINT
DECLARATIVES
DELETE
DEPENDING
DESCENDING
DETAIL
DEVICE
DISPLAY
DIVIDE
DIVISION
DOWN

EDITION
ELSE
END
END-COMPILATION
END-PROGRAM
ENDING
ENTER
ENVIRONMENT
EQUAL
ERROR
EVERY
EXAMINE
EXIT

FD
FILE
FILE-CONTROL
FILE-LIMIT

FILE-LIMITS
FILLER
FINAL
FIRST
FOOTING
FOR
FORWARD
FROM

GENERATE
GIVING
GO
GREATER
GROUP

HEADING
HIGH-VALUE
HIGH-VALUES

I-O
I-O CONTROL
ID
IDENTIFICATION
IF
IN
INDEX
INDEX-BLOCK
INDEXED
INDICATE
INITIATE
INPUT
INPUT-OUTPUT
INSTALLATION
INTO
INVALID
IS

JUST
JUSTIFIED

KEY
KEYS

LABEL
LAST
LEADING
LEFT
LESS
LIMIT
LIMITS

LINE
LINE-COUNTER
LINES
LINKAGE
LOCK
LOW-VALUE
LOW-VALUES

MEMORY
MODE
MODIFICATION-CODE
MODULES
MOVE
MULTIPLE
MULTIPLY

NEGATIVE
NEXT
NO
NOT
NOTE
NUMBER
NUMERIC

OBJECT-COMPUTER
OCCURS
OF
OFF
OMITTED
ON
OPEN
OPTIONAL
OR
OUTPUT

PACKED
PAGE
PAGE-COUNTER
PERFORM
PF
PH
PIC
PICTURE
PLUS
POSITION
POSITIVE
PROCEDURE
PROCEED
PROCESSING

PROGRAM
PROGRAM-ID
PURGE

QUOTE
QUOTES

RANDOM
RD
READ
RECORD
RECORDS
REDEFINES
REEL
RELEASE
REMARKS
RENAMES
REPLACING
REPORT
REPORTING
REPORTS
RERUN
RESERVE
RESET
RETENTION-CYCLE
RETENTION-PERIOD
RETURN
REVERSED
REWIND
REWRITE
RF
RH
RIGHT
ROUNDED
RUN

SAME
SD
SEARCH
SECTION
SECURITY
SEEK
SEGMENT-LIMIT
SELECT
SENTENCE
SEQUENTIAL
SET
SIGN
SIZE
SORT
SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
START
STATUS
STOP
SUBTRACT
SUM
SYNC
SYNCHRONIZED
SYSIN
SYSOUT

TALLY
TALLYING
TAPE
TERMINATE
THAN
THROUGH
THRU
TIMES
TO
TYPE

UNIT
UNTIL
UP
UPON
USAGE
USE
USING

VALUE
VALUES
VARYING

WHEN
WITH

WORDS
WORKING-STORAGE
WRITE

ZERO
ZEROES
ZEROS

D. ANS STANDARD CONTROL CHARACTERS

The control character is passed as the first character of the data record. This character defines the operation on the carriage control tape channel of a printer or the stacker select on a punch. Following is a list of characters and the corresponding operation:

Character	Operation
(blank)	Space one line before printing
0	Space two lines before printing
-	Space three lines before printing
+	Suppress space before printing
1	Skip to channel 1 before printing
2	Skip to channel 2 before printing
3	Skip to channel 3 before printing
4	Skip to channel 4 before printing
5	Skip to channel 5 before printing
6	Skip to channel 6 before printing
7	Skip to channel 7 before printing
8	Skip to channel 8 before printing
9	Skip to channel 9 before printing
A	Skip to channel 10 before printing
B	Skip to channel 11 before printing
C	Skip to channel 12 before printing
V	Select stacker 1
W	Select stacker 2

E. RECORDING MODES

The recording mode is determined by the compiler through a scan of each record description and is not a specification by the user. A discussion of recording mode is provided to give a clearer understanding of the file structure. The recording mode may be F (fixed), V (variable), P (packed), or S (segmented).

RECORDING MODE F

All of the records in a file are the same length and each is wholly contained in one block. Blocks may contain more than one record, and there is a fixed number of records per block.

RECORDING MODE V

The records are variable in length, and each record is wholly contained in one block. Blocks may contain more than one record and the blocks are variable in length (variable length records on tape).

RECORDING MODE P

The records are variable length and each record is wholly contained within a block. The block which has a fixed length, contains a variable number of records. The records are packed back to back from the beginning of the block (variable length records on disc).

RECORDING MODE S

The records are variable length and each record is wholly contained within the block. The block which has a fixed length, is divided in equal size segments. Each segment is the size of the largest record. The block contains one variable length record in each of its segments.

STANDARD SEQUENTIAL FILES

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

F	If all the records are defined as being the same size.
V	If the records are defined variable in size and the file is a tape file.

P If the records are defined as variable in size and the file a mass storage file.

RELATIVE FILES

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

F If all the records are defined as being the same size.

S If the records are defined as variable in size.

Files assigned to unit record devices, and files with indexed organization will be recorded in F mode only.

F. FILE PROCESSING SUMMARY

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

The exact contents of the Environment Division depends on the method used to process files in the COBOL program. Before the language elements used in the Environment Division can be discussed meaningfully, some background in the file processing techniques available to the COBOL user must be given. (A detailed discussion of file processing appears in the **MRX/OS Control Program and Data Management Services, Extended Reference** manual.)

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

DATA ORGANIZATION

Three types of data organization are available to the MRX COBOL user: sequential, relative, and indexed.

The means of creating or retrieving logical records in a file depends on the type of organization used. Each type of data organization is incompatible with the others. Organization of an input file must be the same as the organization of the file when it was created.

Files organized sequentially may contain variable length records.

Files with a relative organization may functionally contain variable length records; the physical allocation however, is fixed length and based on the largest record specified.

Files with an indexed organization must contain fixed length records only.

SEQUENTIAL DATA ORGANIZATION

When sequential data organization is used, the logical records in a file are positioned in an established sequence. The sequence is established as a result of writing the records to the file. Sequential data organization must be used for tape and unit record files and may be used for mass storage files. No key is associated with records in a sequentially organized file.

RELATIVE DATA ORGANIZATION

Relative data organization is characterized by the use of the relative addressing scheme. When this scheme is used, the position of the logical records in a file is determined relative to the first record of the file starting with the initial value of 1.

The relative record address is transformed into a direct block address and a relative position within the block. An ACTUAL KEY containing the value of the record number requested is used to identify randomly accessed records. Files with relative data organization must be assigned to mass storage devices.

INDEXED DATA ORGANIZATION

When indexed data organization is used, the position of each logical record in a file is determined by indexes created with the file and maintained by the system. Up to five keys may be associated with the records in an indexed file; one primary key, and up to four secondary keys. The primary key is required and must have a unique "value" for each record in the file. The secondary keys are optional. It is not required that the primary key is located within the record itself. However, any secondary key specified must be located within the record. The MRX COBOL user may refer to the primary key only, and must consider all secondary keys as data items. The "value" of the primary key is symbolic. Indexed files may be assigned to mass storage devices only.

ACCESS METHODS

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

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

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

ACCESSING A SEQUENTIAL FILE

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

ACCESSING A RELATIVE FILE

A relative file may be accessed either sequentially or randomly. Records may be created, retrieved, or added sequentially or randomly.

SEQUENTIAL ACCESS

The ACTUAL KEY clause is not required when a relative file is accessed sequentially.

When a relative file is created sequentially, the records are written in order, that is, the creation process corresponds exactly to the creation of a sequential file.

When a relative file is being read sequentially, the records are made available in the physical order of the record positions, position 1 through position of last record allocated. All records, including records existing from a previously created file, as a result of a random creation process, are made available.

There is no explicit way of updating a record in a relative file. When a relative file is accessed sequentially, a READ followed by a WRITE is considered an update; the record-position used for the WRITE is the same as the record-position used for the READ. Subsequent WRITE statements will write records into consecutive record positions.

RANDOM ACCESS

When accessing a relative file randomly, the ACTUAL KEY clause is required. The system uses the ACTUAL KEY to determine the relative position of the record to be accessed.

When a relative file is created randomly, records are written into positions specified by the ACTUAL KEY.

To retrieve, or write a record randomly, the ACTUAL KEY must contain the position of the record relative to the beginning of the file. The first position in the file has a value of one.

Since dummy records are not provided by the system, no distinction can be made between the update and the write process. Only the write process exists. (Writing a record may be considered an update if the record is written into a position containing a previously written record, or a user-supplied dummy record.)

ACCESSING AN INDEXED FILE

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

SEQUENTIAL ACCESS

An indexed file may be created sequentially only. When creating an indexed file, the ACTUAL KEY clause must be specified. It is used to supply the value of the primary key to be associated with the record. The key values provided must be unique and in collating sequence. Records will appear in the file in the order in which they are written.

When an indexed file is being read sequentially, the records are made available in the order which logically corresponds to the sequence of the keys.

A FORWARD KEY clause may be specified when records are retrieved sequentially from an indexed file. If specified, the system will place the primary key value of the next record in the file into FORWARD KEY; that is, after a READ, the FORWARD KEY will contain the primary key value of the record that will be obtained on the next read.

To update, add, or delete a record sequentially, the ACTUAL KEY clause is required. The ACTUAL KEY contains the primary key value of the record to be added, updated, or deleted.

When a record is to be added sequentially, the key value specified by ACTUAL KEY must be greater than the key value of the last record read, and less than the key value of the next record in sequence. The FORWARD KEY may be used to check the key value range for a sequential add.

To delete or update a record, the key value specified by ACTUAL KEY must be equal to the key value of the last record read.

RANDOM ACCESS

ACTUAL KEY must be specified and a key value supplied when records are retrieved, updated, added, or deleted.

To update a record randomly, it is required that the record is retrieved first, that is, the execution of a REWRITE statement must logically be preceded by the execution of a READ statement.

For a randomly accessed indexed file, a record is considered "found" when the ACTUAL KEY is equal to the primary key value for the record.

G. INDEX – BLOCK SIZE FOR INDEXED FILES

CALCULATING BY TABLE

The minimum and optimum index block size may be calculated by Tables G-1 through G-4. Refer to the **MRX/OS Control Program and Data Management Services, Extended Reference** manual for the layouts of the index portion of indexed files.

MINIMUM INDEX BLOCK

There is a *minimum index block size* for every indexed file depending on key size and file size. The user may utilize any index block size larger than the minimum, if he has memory space for a larger index block. The larger the index block the better retrieval becomes on random processing. If the user goes below the minimum index block size there is the possibility of not being able to create the file size as planned.

OPTIMUM INDEX BLOCK

When planning the creation of indexed files, the user must decide whether he wants to process the directory-directory, which resides on mass storage, in a main memory buffer. This option speeds up random processing, but requires extra space for the buffer. If the mode of processing is with a main-memory buffer there is well-defined *optimum index block size* which minimizes memory space for the index buffer and directory-directory buffer.

Once the user has determined his mode of processing, Table G-1 is used to determine minimum-keys/block and Table G-2 is used to determine optimum keys/block. Note that in using Table G-1 and Table G-2, the larger of the two values in the file size is the determining factor. Also note that these tables were computed for consistency for maximum key size and one million records as the upper limit. There will be some index block sizes generated that exceed one track in number of bytes. This exceeds the system limit for block sizes. The user will have to choose a smaller key size or smaller file size.

PROGRAMMING CONSIDERATIONS

The keys/block is entered in the Control Language //DEFINE statement along with key size. The corresponding minimum or optimum index block size can be calculated from Table G-3. The resulting index block size is then entered in the COBOL source program via the INDEX-BLOCK Clause.

If the user has determined to calculate the optimum keys/block and optimum index block size, Table G-4 is used to calculate the number of bytes for the main-memory buffer for the directory-directory entries.

The user must be careful not to exceed the file maximum at creation time when using the optimum block size – when he utilizes the main-memory buffer to hold the directory-directory entries for random processing, the buffer would not be able to hold all the entries, thus writing over the user program. Thus, when choosing an index block size other than the optimum and the main-memory buffer is used to process the directory-directory entries, the buffer size should be the size of the index block, as the system checks for overflow at creation time.

EXAMPLES

The following examples illustrate how to calculate the minimum index block size, the larger than minimum index block size, and the optimum index block size.

MINIMUM INDEX BLOCK SIZE

The minimum index block size can be calculated with the following steps.

1. In Table F-1 locate the number of records in the file and the key size. For example, if the number of records is 20,000 and the key size is 10, the minimum keys per block is 24.
2. The keys/block is entered in the Control Language //DEFINE statement along with the key size.
3. The corresponding minimum index block size is calculated from Table F-3 using the minimum block size formula. For this example with minimum keys/block of 24 and key size of 10; the minimum index block size is 384 bytes.
4. The minimum index block size is then entered into the source program.

LARGER THAN MINIMUM INDEX BLOCK SIZE

Similar to the minimum index block size, an index block that is larger than the minimum may be calculated with the same steps. The difference is found in estimating the keys/block; it must be greater than or equal to the minimum keys/block selected.

OPTIMUM INDEX BLOCK SIZE

The optimum index block size can be calculated with the following steps.

1. In Table F-2 locate the number of records in the file and the key size. For example, if the number of records is 20,000 and the key size is 10, the optimum keys per block is 30.

2. The keys/block is entered in the Control Language //DEFINE statement along with the key size.

3. The corresponding optimum index block size is calculated from Table F-3 using the optimum block size formula. For this example with optimum keys/block of 30 and key size of 30, the optimum index block size is 475 bytes.

4. The optimum index block size is then entered into the source program.

5. For optimum keys/block and optimum index block size, Table F-4 is used to calculate the main-storage buffer for the directory to the directory entries. For this example, the number of bytes required for the buffer for the directory to the directory is 250.

Table G-1. Minimum Keys/Block

Records in File	Key Size in Bytes														
	2	3	4	5	6	7	8	9	10	11 to 15	16 to 20	21 to 25	26 to 35	36 to 50	51 to 100
0 - 5,000	13	14	14	14	15	15	15	15	15	16	16	16	16	17	17
5,000 - 10,000	16	17	18	18	18	19	19	19	19	20	20	20	21	21	21
10,000 - 15,000	19	20	20	21	21	21	22	22	22	23	23	23	23	24	24
15,000 - 20,000	20	21	22	23	23	23	24	24	24	25	25	26	26	26	26
20,000 - 25,000	22	23	24	24	25	25	25	26	26	27	27	27	28	28	28
25,000 - 30,000	23	24	25	26	26	27	27	27	28	28	29	29	29	30	30
30,000 - 35,000	25	26	27	27	28	28	28	29	29	30	30	31	31	31	32
35,000 - 40,000	26	27	28	28	29	29	30	30	30	31	32	32	32	33	33
40,000 - 45,000	27	28	29	30	30	31	31	31	31	32	33	33	34	34	34
45,000 - 50,000	28	29	30	31	31	32	32	32	33	34	34	34	35	35	36
50,000 - 60,000	29	31	32	32	33	34	34	34	35	36	36	37	37	37	38
60,000 - 70,000	31	32	34	34	35	35	36	36	36	37	38	38	39	39	40
70,000 - 80,000	32	34	35	36	36	37	37	38	38	39	40	40	41	41	42
80,000 - 90,000	34	35	36	37	38	38	39	39	40	41	41	42	42	43	43
90,000 - 100,000	35	36	37	38	39	40	40	41	41	42	43	43	44	44	45
100,000 - 125,000	37	39	40	41	42	43	43	44	44	45	46	47	47	48	48
125,000 - 150,000	40	41	43	44	45	45	46	46	47	48	49	49	50	51	51
150,000 - 175,000	42	44	45	46	47	48	48	49	49	50	51	52	53	53	54
175,000 - 200,000	44	46	47	48	49	50	50	51	51	53	54	54	55	56	56
200,000 - 250,000	47	49	51	52	53	54	54	55	55	57	58	58	59	60	61
250,000 - 300,000	50	52	54	55	56	57	58	58	59	61	61	62	63	64	64
300,000 - 350,000	52	55	57	58	59	60	61	62	62	64	65	65	66	67	68
350,000 - 400,000	55	57	59	61	62	63	63	64	65	67	68	68	69	70	71
400,000 - 450,000	57	60	62	63	64	65	66	67	67	69	70	71	72	73	74
450,000 - 500,000	59	62	64	65	67	68	68	69	70	72	73	74	74	75	76
500,000 - 600,000	63	66	68	69	71	72	73	73	74	76	77	78	79	80	81
600,000 - 700,000	66	69	71	73	74	75	76	77	78	80	81	82	83	84	85
700,000 - 800,000	69	72	74	76	78	79	80	81	81	84	85	86	87	88	89
800,000 - 900,000	72	75	77	79	81	82	83	84	85	87	88	89	91	91	93
900,000 - 1,000,000	74	78	80	82	84	85	86	87	88	90	92	93	94	95	96

Table G-2. Optimum Keys/Block

Records in File	Key Size in Bytes													
	2	3	4	5	6	7	8	9	10	11 to 15	16 to 20	21 to 25	26 to 35	36 to 50	51 to 100
0 - 5,000	16	17	18	18	18	19	19	19	19	20	20	20	21	21	21
5,000 - 10,000	20	21	22	23	23	23	24	24	24	25	25	25	26	26	26
10,000 - 15,000	23	24	25	26	26	27	27	27	27	28	29	29	29	30	30
15,000 - 20,000	26	27	28	28	29	29	29	30	30	31	32	32	32	33	33
20,000 - 25,000	28	29	30	31	31	32	32	32	33	34	34	34	35	35	36
25,000 - 30,000	29	31	32	32	33	34	34	34	35	36	36	37	37	37	38
30,000 - 35,000	31	32	33	34	35	35	36	36	37	37	38	38	39	39	40
35,000 - 40,000	32	34	35	36	36	37	37	38	38	39	40	40	41	41	42
40,000 - 45,000	34	35	36	37	38	38	39	39	40	41	41	42	42	43	43
45,000 - 50,000	35	36	37	38	39	40	40	41	41	42	43	43	44	44	45
50,000 - 60,000	37	39	40	41	42	42	43	43	43	45	45	46	46	47	48
60,000 - 70,000	39	41	42	43	44	44	45	45	46	47	48	48	49	49	50
70,000 - 80,000	40	42	44	45	46	46	47	47	48	49	50	50	51	52	52
80,000 - 90,000	42	44	45	47	47	48	49	49	50	51	52	52	53	54	54
90,000 - 100,000	44	46	47	48	49	50	50	51	51	53	54	54	55	56	56
100,000 - 125,000	47	49	51	52	53	54	54	55	55	57	58	58	59	60	61
125,000 - 150,000	50	52	54	55	56	57	58	58	59	61	61	62	63	64	64
150,000 - 175,000	52	55	57	58	59	60	61	61	62	64	65	65	66	67	68
175,000 - 200,000	55	57	59	61	62	63	63	64	65	67	68	68	69	70	71
200,000 - 250,000	59	62	64	65	67	68	68	69	70	72	73	74	74	75	76
250,000 - 300,000	63	66	68	69	71	72	73	73	74	76	77	78	79	80	81
300,000 - 350,000	66	69	71	73	74	75	76	77	78	80	81	82	83	84	85
350,000 - 400,000	69	72	74	76	78	79	80	81	81	84	85	86	87	88	89
400,000 - 450,000	72	75	77	79	81	82	83	84	85	87	88	89	91	91	93
450,000 - 500,000	74	78	80	82	84	85	86	87	88	90	92	93	94	95	96
500,000 - 600,000	79	82	84	87	89	90	91	92	93	96	97	98	100	101	102
600,000 - 700,000	83	87	90	92	94	95	96	97	98	101	102	103	105	106	107
700,000 - 800,000	87	91	94	96	98	99	100	102	102	105	107	108	110	111	112
800,000 - 900,000	90	94	97	100	102	103	105	106	106	110	111	112	114	115	116
900,000 - 1,000,000	93	98	101	103	105	107	108	109	110	113	115	116	118	120	121

Table G-3. Optimum or Minimum Index Block Size

Optimum block size $= 10 + \left\{ \frac{(10)\ (OKB)\ (KS+4)}{9} \right\}$

OKB = Optimum keys/block

KS = Key size

Minimum block size $= 10 + \left\{ \frac{(10)\ (MKB)\ (KS+4)}{9} \right\}$

MKB = Minimum keys/block

KS = Key size

NOTE: $\{\ \}$ = Round up if result not whole integer.

Table G-4. Bytes Required in Buffer for Directory-Directory Entries

Usage $= \left[\frac{9(OBS-10)}{10} \right] = US$

Number keys/primary index block $= \left[\frac{US}{KS+4} \right] = NKP$

Number keys/directory block $= \left[\frac{US}{KS+2} \right] = NKD$

Total number keys represented/
directory block $= (NKP)\ (NKD) = NKRD$

Number entries in
Directory-directory block $= \left\{ \frac{\text{file size}}{NKRD} \right\} = NKDD$

Number of bytes required for
buffer for directory-directory
entries $= 10 + (KS+2)\ (NKDD)$

NOTE: $\{\ \}$ = Round up if result not whole integer.

$[\ ]$ = Round down if result not whole integer.

CALCULATING BY FORMULA

If the user wishes to calculate keys/block based on a different file maximum than given in Tables G-1 and G-2, the following algorithms, along with Table G-5, can be used to compute minimum and optimum keys/block. The constants Ko and Km are taken from Table G-5 based on key size.

$$\text{Optimum (OKB)} = \left\{ \sqrt[3]{\frac{FS}{Ko}} \right\}$$

$$\text{Minimum (MKB)} = \left\{ \sqrt[3]{\frac{FS}{Km}} \right\}$$

FS = Maximum File Size

NOTE: $\{\ \}$ = Round up if result not whole integer.

Table G-5. Constants for Alternate Algorithm

KS	K_o	K_m
2	1.2500	2.5000
3	1.0888	2.1776
4	.9877	1.9753
5	.9184	1.8367
6	.8681	1.7361
7	.8299	1.6598
8	.8000	1.6000
9	.7759	1.5518
10	.7562	1.5124
11	.7396	1.4792
12	.7256	1.4512
13	.7136	1.4272
14	.7031	1.4062
15	.6940	1.3880
16	.6859	1.3718
17	.6787	1.3574
18	.6722	1.3444
19	.6664	1.3328
20	.6612	1.3224
21	.6564	1.3128
22	.6520	1.3040
23	.6480	1.2960
24	.6443	1.2886
25	.6409	1.2818
26	.6378	1.2756
27	.6348	1.2696
28	.6321	1.2642
29	.6296	1.2592
30	.6272	1.2544
31	.6249	1.2498
32	.6228	1.2456
33	.6209	1.2418
34	.6190	1.2380
35	.6172	1.2344
36	.6156	1.2312
37	.6140	1.2280
38	.6125	1.2250
39	.6111	1.2222
40	.6097	1.2194
41	.6084	1.2168
42	.6072	1.2144
43	.6060	1.2120
44	.6049	1.2098
45	.6038	1.2076
46	.6028	1.2056
47	.6018	1.2036
48	.6009	1.2018
49	.6000	1.2000
50	.5991	1.1982
51	.5983	1.1966
52	.5975	1.1950
53	.5967	1.1934
54	.5959	1.1918
55	.5952	1.1904
56	.5945	1.1890
57	.5939	1.1878
58	.5932	1.1864
59	.5926	1.1852
60	.5920	1.1840
61	.5914	1.1828
62	.5908	1.1816
63	.5903	1.1806
64	.5897	1.1794
65	.5892	1.1784
66	.5887	1.1774
67	.5882	1.1764
68	.5878	1.1756
69	.5873	1.1746
70	.5868	1.1736
71	.5864	1.1728
72	.5860	1.1720
73	.5856	1.1712
74	.5852	1.1704
75	.5848	1.1696
76	.5844	1.1688
77	.5840	1.1680
78	.5837	1.1674
79	.5833	1.1666
80	.5830	1.1660
81	.5827	1.1654
82	.5823	1.1646
83	.5820	1.1640
84	.5817	1.1634
85	.5814	1.1628
86	.5811	1.1622
87	.5808	1.1616
88	.5805	1.1610
89	.5802	1.1604
90	.5800	1.1600
91	.5797	1.1594
92	.5794	1.1588
93	.5792	1.1584
94	.5789	1.1578
95	.5787	1.1574
96	.5785	1.1570
97	.5782	1.1564
98	.5780	1.1560
99	.5778	1.1556
100	.5776	1.1552

H. COBOL ERROR MESSAGES

The COBOL compiler issues two types of error messages. They are source error diagnostic messages that are printed at the end of the COBOL compilation listing and object time error messages that are listed on the SYSOUT file.

COBOL SOURCE LISTING ERROR MESSAGES

The COBOL source error messages are printed at the end of the compilation section of the COBOL listing. These messages are of two types: COBOL compiler errors, and errors made by the COBOL programmer. In the event that a COBOL compiler error occurs, contact a systems engineer.

COBOL COMPILER ERRORS

Only three diagnostic messages are issued by the compiler to warn the user of errors within the COBOL compiler. These messages are listed below.

ERROR CODE	MESSAGE TEXT
CB0X001	ERROR IN INPUT RECORD (CHECK PHASE AND ERROR NBRS).
CB0X002	MESSAGE IS NOT AVAILABLE FOR THIS PASS OR PHASE.
CB0X003	*** COMPILER ERROR: CODE=ppnn ***

where:

pp	is a one or two digit pass number
nn	specifies the compiler error condition within the pass

These codes do not appear in this manual. They are for MRX internal use only.

COBOL PROGRAMMING ERRORS

COBOL programming errors are printed at the end of the COBOL source listing. The messages have the following format:

LINE NUMBER	ERROR TYPE	ERROR CODE	CLAUSE NAME	MESSAGE TEXT
nnnn	t	aappnnn	aaaaaaaa. . .a	text $\left\{\begin{array}{c}\text{*global insert*}\\ \text{predetermined insert}\end{array}\right\}$ text

where:

nnnn	is a 4-digit decimal number specifying the line in the source listing where the error occurred. For //PAR statements, nnnn is the number of the //PAR statement in the SYSOUT file.
t	is variously W, F, or U designating the type of error as warning, fatal, or USASI.
aappnnn	is a 7-character error code where aa is always CB specifying the COBOL compiler as the source of the error, pp is variously 01 through 09 specifying the pass within the compiler, and nnn is a 3-digit decimal number specifying the error within the pass.
aaa. . .a	is a clause name of 1 to 14 alphanumeric characters specifying the clause in error. Not all messages are preceded by a clause entry.
text $\left\{\begin{array}{c}\text{*global insert*}\\ \text{predetermined insert}\end{array}\right\}$ text	is the text of the message. Some messages have words or phrases inserted in them at the time the error occurs. The inserted text is preceded and followed by a single asterisk. In messages having global variable inserts, a series of blank spaces preceded and followed by an asterisk is shown in the list of messages. In messages having specific, predetermined variables, the variables are listed one below the other and are preceded and followed by braces.

LINE NUMBER	ERROR TYPE	ERROR CODE	CLAUSE NAME	MESSAGE TEXT
nnnn	F	CB01001		FORMAT ERROR -- CONTINUATION CANNOT START PRIOR TO B MARGIN.
nnnn	F	CB01002		CONTINUATION NOT VALID -- CONTINUATION IGNORED.
nnnn	F	CB01003		DELIMITER NOT FOLLOWED BY SPACE -- SPACE ASSUMED.
nnnn	F	CB01004		INCORRECT SUBSCRIPT. Left parenthesis must be followed either by a data-name or a numeric literal.
nnnn	F	CB01005		ILLEGAL ELEMENT. An illegal element is any combination of characters that is not legal in COBOL.
nnnn	U	CB01006		';' NOT IMPLEMENTED IN THIS LEVEL OF COMPILER.
nnnn	F	CB01007		NON-COBOL CHARACTER NOT WITHIN QUOTES -- ELEMENT DROPPED.
nnnn	F	CB01008		OPENING QUOTES NOT PRECEDED BY A SPACE -- PREVIOUS ELEMENT DROPPED.
nnnn	F	CB01009		DELIMITER MISSING. A delimiter, such as a comma, is missing in a subscript. Or the delimiting apostrophe of an alphanumeric literal is missing.
nnnn	F	CB01010		NONINTEGER NUMERIC LITERAL USED AS INDEX. A numeric literal used as a subscript must be an unsigned integer.
nnnn	F	CB01011		END COMPILE CARD MISSING.
nnnn	F	CB01012		DUPLICATE LEFT PARENTHESIS -- SECOND ONE DROPPED.
nnnn	U	CB01013		BLANK CANNOT FOLLOW LEFT PARENTHESIS.
nnnn	F	CB01014		ALPHA LITERAL EXCEEDS 120 -- EXCESS IS TRUNCATED.
nnnn	F	CB01015		ALPHA LITERAL NOT TERMINATED CORRECTLY.
nnnn	F	CB01016		ALPHA LITERAL WITH NO DATA.
nnnn	F	CB01017		NUMERIC LITERAL EXCEEDS 18 DIGITS.

LINE NUMBER	ERROR TYPE	ERROR CODE	CLAUSE NAME	MESSAGE TEXT
nnnn	F	CB01018		PROGRAMMER-ASSIGNED WORD EXCEEDS 30 CHARACTERS.
nnnn	F	CB01019		* * IS ILLEGAL IN THIS DIVISION.
nnnn	F	CB01020		* * IS RESERVED FOR A HIGHER LEVEL COMPILER.
nnnn	U	CB01021		TERMINATING PERIOD MISSING ON LAST CARD.
nnnn	W	CB01022		//PAR ERROR -- INVALID DELIMITER IN COLUMN * *. nnnn is the //PAR statement in the SYSOUT file.
nnnn	F	CB01023		//PAR ERROR -- UNDEFINED KEYWORD IN COLUMN * *. nnnn is the //PAR statement in the SYSOUT file.
nnnn	W	CB01024		//PAR ERROR -- R MARGIN MUST BE BETWEEN 41-120, COLUMN * *. nnnn is the //PAR statement in the SYSOUT file.
nnnn	U	CB01025		BLANK CANNOT PRECEDE A RIGHT PARENTHESIS.
nnnn	F	CB01026		FORMAT ERROR -- EXPECT DATA IN MARGIN B.
nnnn	F	CB01027		EXPECTING IDENTIFICATION DIVISION -- FOUND * *.
nnnn	F	CB01028		EXPECTING KEYWORD -- FOUND * *.
nnnn	F	CB01029		IDENTIFICATION DIVISION MISSING OR OUT OF SEQ.
nnnn	F	CB01030		DIVISION HEADER DUPLICATE OR OUT OF SEQ.
nnnn	F	CB01031		'ENVIRONMENT DIVISION.' HEADER MISSING OR OUT OF SEQ -- ASSUMED TO EXIST HERE.
nnnn	F	CB01032		KEYWORD EXPECTED IN MARGIN A.
nnnn	F	CB01033		DIVISION HEADERS MUST END WITH 'DIVISION.'.
nnnn	U	CB01034		PARAGRAPH HEADER MUST BE IMMEDIATELY TERMINATED BY A PERIOD.
nnnn	U	CB01035		PARAGRAPH HEADER DUPLICATE OR OUT OF SEQ.

LINE NUMBER	ERROR TYPE	ERROR CODE	CLAUSE NAME	MESSAGE TEXT
nnnn	F	CB01036		INVALID PROGRAM-ID. The PROGRAM-ID does not adhere to the rules for PROGRAM-ID's.

LINE NUMBER	ERROR TYPE	ERROR CODE	CLAUSE NAME	MESSAGE TEXT
nnnn	F	CB01040		OPENING QUOTE MISSING -- ASSUMED TO EXIST.
nnnn	F	CB01041		'PROCEDURE DIVISION.' HEADER MISSING OR OUT OF SEQ -- ASSUMED TO EXIST HERE.
nnnn	F	CB01042		ALL SECTION HEADERS MUST END WITH 'SECTION.'.
nnnn	F	CB01043		PICTURE STRING MISSING.
nnnn	F	CB01044 CB01045		'DATA DIVISION.' HEADER MISSING OR OUT OF SEQ -- ASSUMED TO EXIST HERE.
nnnn	F	CB01045		PICTURE STRING EXCEEDS 30 CHARACTERS.
nnnn	F	CB01046		PROGRAM-ID INVALID OR MISSING. The PROGRAM-ID is either missing or has previously been determined invalid by message CB01036.
nnnn	U	CB01048		PUNCTUATION MARKS MUST NOT BE PRECEDED BY A SPACE.
nnnn	F	CB01049		* * IS ILLEGAL IN CONTEXT.
nnnn	F	CB01050		PERIOD MISSING.
nnnn	W	CB01051		//PAR ERROR -- INVALID NUMERIC, COLUMN * *.
nnnn	F	CB01052		//PAR ERROR -- INVALID MEMBER NAME, COLUMN * *.
nnnn	F	CB01053		//PAR ERROR -- MISSING MEMBER NAME, COLUMN * *.
nnnn	F	CB01054		INVALID NUMERIC LITERAL * *.
nnnn	U	CB02001		HEADERS MUST BEGIN IN AREA A.
nnnn	U	CB02002		FORMAT ERROR -- ILLEGAL ELEMENT IN COLUMNS 8-11.
nnnn	U	CB02004		COMMA NOT ALLOWED AS PUNCTUATION THIS LEVEL COBOL.
nnnn	F	CB02006		EXPECTING KEY ELEMENT -- FOUND * *.
nnnn	W	CB02007		NUGATORY OR MISPLACED TERMINAL PERIOD.

LINE NUMBER	ERROR TYPE	ERROR CODE	CLAUSE NAME	MESSAGE TEXT
nnnn	U	CB02008		'INPUT-OUTPUT SECTION.' HEADER MISSING.
nnnn	U	CB02009		'FILE-CONTROL.' HEADER MISSING.
nnnn	U	CB02010		'INPUT-OUTPUT SECTION.' AND 'FILE-CONTROL.' HEADERS MISSING.
nnnn	U	CB02011		CLAUSE OUT OF ORDER.
nnnn	F	CB02012		HEADER DUPLICATE OR OUT OF ORDER.
nnnn	F	CB02013		SELECT CLAUSE MISSING.
nnnn	U	CB02014		FILE-CONTROL ENTRY MISSING.
nnnn	F	CB02015		ASSIGN CLAUSE MISSING, ILLEGAL, OR OUT OF ORDER.
nnnn	U	CB02016		TERMINAL PERIOD MISSING.
nnnn	U	CB02017		'I-O-CONTROL.' HEADER MISSING.
nnnn	F	CB02018		'INPUT-OUTPUT SECTION.' AND 'FILE-CONTROL.' HEADERS AND SELECT CLAUSE MISSING.
nnnn	U	CB02019		'CONFIGURATION SECTION.' HEADER MISSING.
nnnn	F	CB02020		INVALID NUMERIC LITERAL -- CHECK DECIMAL-POINT IS COMMA CLAUSE.
nnnn	F	CB02021		IMPLEMENTOR NAME OUT OF ORDER -- NOT PROCESSED.
nnnn	F	CB02022		CLAUSE OUT OF ORDER -- CLAUSE DROPPED.
nnnn	U	CB02023		'SPECIAL-NAMES.' HEADER MISSING.
nnnn	U	CB02024		'CONFIGURATION SECTION.' HEADER MISSING -- ATTEMPT PROCESSING SELECT CLAUSE.
nnnn	U	CB02025		CONFIGURATION SECTION ENTRY MISSING.
nnnn	F	CB02026		DUPLICATE CLAUSE * *.
nnnn	F	CB02028		IMPLEMENTOR NAME ILLEGAL OR MISSING.
nnnn	F	CB02030		ILLEGAL OR MISPLACED ELEMENT * *.

LINE NUMBER	ERROR TYPE	ERROR CODE	CLAUSE NAME	MESSAGE TEXT
nnnn	U	CB02031		'SOURCE-COMPUTER.' HEADER ILLEGAL OR MISSING.
nnnn	U	CB02032		SOURCE-COMPUTER NAME ILLEGAL OR MISSING.
nnnn	U	CB02033		'OBJECT-COMPUTER.' HEADER ILLEGAL OR MISSING.
nnnn	U	CB02034		OBJECT-COMPUTER NAME ILLEGAL OR MISSING.
nnnn	F	CB02035		INCOMPLETE CLAUSE * *.
nnnn	F	CB02036		NUMERIC LITERAL MUST BE AN UN-SIGNED INTEGER.
nnnn	F	CB02037		DATA NAME CANNOT BE SUBSCRIPTED.
nnnn	F	CB02038		ILLEGAL LENGTH OF CURRENCY-SIGN LITERAL.
nnnn	F	CB02039		ILLEGAL CHARACTER USED AS CURRENCY-SIGN LITERAL.
nnnn	F	CB02040		MEMORY SIZE MUST BE UNSIGNED LITERAL.
nnnn	F	CB02041		SEGMENT-LIMIT MUST BE AN UNSIGNED INTEGER IN THE RANGE OF 1-49.
nnnn	F	CB02042		PREVIOUS CLAUSE INCOMPLETE.
nnnn	U	CB02044		'CONFIGURATION SECTION.' AND 'INPUT-OUTPUT SECTION.' HEADERS MISSING.
nnnn	F	CB02045		REQUIRED ELEMENT MISSING -- FOUND * *.
nnnn	U	CB02046		TERMINAL PERIOD MISSING ON PARA-GRAPH AND/OR SECTION NAME.
nnnn	U	CB02047		FIRST SECTION OF PROCEDURE DIVISION MISSING -- IS REQUIRED.
nnnn	U	CB02048		LEVEL INDICATOR MUST START IN COLUMNS 8-11.
nnnn	U	CB02049		TERMINAL PERIOD OF PRECEDING STATEMENT MISSING. This error applies to the Environment Division.

LINE NUMBER	ERROR TYPE	ERROR CODE	CLAUSE NAME	MESSAGE TEXT
nnnn	F	CB02050		ILLEGAL OR MISSING ELEMENT PRECEDING * *.
nnnn	U	CB02051		'CONFIGURATION SECTION.' AND 'SOURCE-COMPUTER.' HEADERS MISSING.
nnnn	U	CB02052		'CONFIGURATION SECTION.', 'SOURCE-COMPUTER.' AND 'OBJECT-COMPUTER.' HEADERS MISSING.
nnnn	U	CB02053		'OBJECT-COMPUTER.' AND 'INPUT-OUTPUT SECTION.' HEADERS MISSING.
nnnn	U	CB02054		'OBJECT-COMPUTER.' AND 'SPECIAL-NAMES.' HEADERS MISSING.
nnnn	F	CB02055		ILLEGAL ELEMENT IN STATEMENT * *.
nnnn	U	CB02056		FORMAT ERROR -- ILLEGAL ELEMENT IN COLUMNS 8-11.
nnnn	U	CB02059		REDEFINES CLAUSE MUST BE FIRST IN DATA DESCRIPTION.
nnnn	F	CB02060		ILLEGAL LEVEL NUMBER -- TREATED AS LEVEL 50.
nnnn	U	CB02061		'FILE SECTION.' HEADER MISSING.
nnnn	F	CB02062		ILLEGAL CLAUSE * *.
nnnn	U	CB02063		SECTION HEADER MISSING -- WORKING-STORAGE ASSUMED.
nnnn	F	CB02064		RECORD DESCRIPTION FOR LAST FD ENTRY MISSING.
nnnn	F	CB02066		DATA ITEM DESCRIPTION ENTRY NOT ALLOWED UNDER AN FD-ENTRY.
nnnn	F	CB02067		LAST FD ENTRY INCOMPLETE.
nnnn	F	CB02068		FD CLAUSE NOT ALLOWED IN RECORD DESCRIPTION ENTRY.
nnnn	F	CB02069		FILE-NAME ILLEGAL OR MISSING.
nnnn	F	CB02070		FD LEVEL INDICATOR MISSING.
nnnn	F	CB02071		ILLEGAL LEVEL NUMBER IN RECORD DESCRIPTION ENTRY.
nnnn	F	CB02072		LEVEL NUMBER MISSING -- RECORD DESCRIPTION ENTRY NOT PROCESSED.

LINE NUMBER	ERROR TYPE	ERROR CODE	CLAUSE NAME	MESSAGE TEXT
nnnn	F	CB02074		FD ENTRY ILLEGAL OUTSIDE FILE SECTION -- FILE DESCRIPTION NOT PROCESSED.
nnnn	F	CB02075		FD CLAUSE MISPLACED -- SKIP TO NEXT CONTROL.
nnnn	F	CB02077		LABEL CLAUSE OF FD ENTRY MISSING.
nnnn	F	CB02078		EXPECTED DATA-NAME -- FOUND * *.
nnnn	F	CB02079		'FILE SECTION.' HEADER DUPLICATED -- HEADER IGNORED.
nnnn	F	CB02080		'FILE SECTION.' HEADER DUPLICATED -- WORKING-STORAGE ASSUMED.
nnnn	F	CB02081		FILE SECTION OUT OF ORDER -- MUST BE FIRST SECTION OF DATA DIVISION.
nnnn	F	CB02082		FILE SECTION DUPLICATED -- SECTION IGNORED.
nnnn	F	CB02084		'WORKING-STORAGE SECTION.' HEADER DUPLICATED -- HEADER IGNORED.
nnnn	F	CB02085		WORKING-STORAGE SECTION MUST PRECEDE LINKAGE SECTION.
nnnn	F	CB02086		WORKING-STORAGE SECTION DUPLICATED -- SECTION IGNORED.
nnnn	F	CB02087		'LINKAGE SECTION.' HEADER DUPLICATED -- HEADER IGNORED.
nnnn	F	CB02088		DUPLICATE IMPLEMENTOR NAME * *.
nnnn	F	CB02089		PICTURE STRING ERROR AT CHARACTER POSITION * *.
nnnn	F	CB02092		THE PICTURE HAS NO DATA LENGTH.
nnnn	F	CB02093		DATA LENGTH OF A NUMERIC CANNOT EXCEED 18 DIGITS.
nnnn	F	CB02094		DATA LENGTH OF AN ALPHA CANNOT EXCEED 16383 BYTES.
nnnn	F	CB02095		ELEMENT LENGTH OF EDITED DATA CANNOT EXCEED 144 BYTES.
nnnn	F	CB02101		TERMINAL PERIOD AND PARAGRAPH NAME MISSING.
nnnn	F	CB02102		NUMERIC PARAGRAPH NAME MUST BE AN UNSIGNED INTEGER.

LINE NUMBER	ERROR TYPE	ERROR CODE	CLAUSE NAME	MESSAGE TEXT
nnnn	F	CB02103		PRIORITY NUMBER MUST BE AN UNSIGNED INTEGER LESS THAN 100.
nnnn	F	CB02104		PARAGRAPH NAME MISSING. EXIT statement is not followed by a paragraph. EXIT must be only statement in paragraph.
nnnn	F	CB02105		ILLEGAL ELEMENT IN SUBSCRIPT LIST -- FOUND * *.
nnnn	F	CB02107		INCOMPLETE SUBSCRIPT LIST -- FOUND * *.
nnnn	F	CB02108		NUMERIC LITERAL USED IN SUBSCRIPT MUST BE AN UNSIGNED INTEGER.
nnnn	F	CB02109		TOO MANY SUBSCRIPT ELEMENTS.
nnnn	F	CB02110		ILLEGAL ARITHMETIC OPERATOR IN SUBSCRIPT * *.
nnnn	F	CB02111		RIGHT PAREN MISSING ON SUBSCRIPT.
nnnn	F	CB02112		IF STATEMENT MUST BE FOLLOWED BY AN ELSE STATEMENT OR A TERMINAL PERIOD.
nnnn	F	CB02113		ELSE STATEMENT DOES NOT HAVE A CORRESPONDING IF.
nnnn	F	CB02114		ILLEGAL USE OF NESTED IF STATEMENT.
nnnn	F	CB02116		PRIORITY-NUMBER MUST NOT EXCEED 2 DIGITS.
nnnn	W	CB03001		* * IN DATA RECORDS CLAUSE NOT DEFINED AT LEVEL 01.
nnnn	F	CB03002		FILE-NAME * * REFERENCED AS DATA-NAME.
nnnn	F	CB03003		INVALID SYSTEM-NAME * *.
nnnn	U	CB03004		FILE-NAME * * REFERENCED AS PROCEDURE-NAME.
nnnn	F	CB03005		FILE-NAME * * REFERENCED AS MNEMONIC-NAME.
nnnn	F	CB03006		NUMERIC LITERAL * * TOO SMALL.
nnnn	F	CB03007		NUMERIC LITERAL * * TOO LARGE.
nnnn	F	CB03008		LITERAL * * MUST BE AN UNSIGNED INTEGER.

LINE NUMBER	ERROR TYPE	ERROR CODE	CLAUSE NAME	MESSAGE TEXT
nnnn	F	CB03009		MISSING NUMERIC LITERAL.
nnnn	W	CB03010		DUPLICATE DATA-RECORD NAME * *.
nnnn	F	CB03011		INDEX-NAME CANNOT BE SUBSCRIPTED -- SUBSCRIPT DROPPED.
nnnn	F	CB03012		DATA-NAME * * REFERENCED AS FILE-NAME.
nnnn	F	CB03013		INDEXING CANNOT BE MIXED WITH SUBSCRIPTING.
nnnn	F	CB03014		ILLEGAL RELATIVE SUBSCRIPT.
nnnn	F	CB03015		MORE THAN 255 INDEXES DEFINED.
nnnn	U	CB03016		DATA-NAME * * REFERENCED AS PROCEDURE-NAME.
nnnn	F	CB03017		DATA-NAME * * REFERENCED AS MNEMONIC-NAME.
nnnn	F	CB03018		INDEX-NAME * * REFERENCED AS FILE-NAME.
nnnn	U	CB03019		INDEX-NAME * * REFERENCED AS PROCEDURE-NAME.
nnnn	F	CB03020		INDEX-NAME * * REFERENCED AS MNEMONIC-NAME.
nnnn	F	CB03021		MNEMONIC-NAME * * REFERENCED AS FILE-NAME.
nnnn	F	CB03022		MNEMONIC-NAME * * REFERENCED AS DATA-NAME.
nnnn	U	CB03023		MNEMONIC-NAME * * REFERENCED AS PROCEDURE-NAME.
nnnn	F	CB03024		REFERENCE TO UNDEFINED FILE-NAME * *.
nnnn	F	CB03025		REFERENCE TO UNDEFINED DATA-NAME * *.
nnnn	F	CB03026		REFERENCE TO UNDEFINED MNEMONIC-NAME * *.
nnnn	F	CB03027		MULTIPLE DEFINITION OF * * FIRST DEFINITION USED.
nnnn	F	CB03028		INDEX-NAME * * NOT A VALID FIRST SUBSCRIPT.

LINE NUMBER	ERROR TYPE	ERROR CODE	CLAUSE NAME	MESSAGE TEXT
nnnn	F	CB03029		INDEX NAME * * NOT A VALID SECOND SUBSCRIPT.
nnnn	F	CB03030		INDEX-NAME * * NOT A VALID THIRD SUBSCRIPT.
nnnn	F	CB03031		INDEX-NAME * * REFERENCED AS DATA-NAME.
nnnn	F	CB03032		DATA-NAME * * ALSO REFERENCED AS PROCEDURE.
nnnn	U	CB03033		INDEX-NAME * * ALSO DEFINED AS PROCEDURE.
nnnn	U	CB03034		MNEMONIC-NAME * * ALSO DEFINED AS PROCEDURE.
nnnn	U	CB03035		FILE-NAME * * ALSO DEFINED AS PROCEDURE.
nnnn	F	CB04001		RELATIVE ORGANIZATION ILLEGAL WITH * $\left\{\begin{array}{l}\text{UNIT RECORD DEVICE.}\\\text{MAGNETIC TAPE DEVICE.}\end{array}\right\}$ *.
nnnn	F	CB04002		INDEXED ORGANIZATION ILLEGAL WITH * $\left\{\begin{array}{l}\text{UNIT RECORD DEVICE.}\\\text{MAGNETIC TAPE DEVICE.}\end{array}\right\}$ *.
nnnn	W	CB04003		MULTIPLE REEL INCONSISTENT WITH * $\left\{\begin{array}{l}\text{MASS STORAGE DEVICE.}\\\text{UNIT RECORD DEVICE.}\end{array}\right\}$ *.
nnnn	W	CB04004		MULTIPLE UNIT INCONSISTENT WITH * $\left\{\begin{array}{l}\text{MAGNETIC TAPE DEVICE.}\\\text{UNIT RECORD DEVICE.}\end{array}\right\}$ *.
nnnn	F	CB04006		ACCESS MODE RANDOM IS ILLEGAL WITH * $\left\{\begin{array}{l}\text{MAGNETIC TAPE DEVICE.}\\\text{UNIT RECORD DEVICE.}\\\text{SEQUENTIAL ORGANIZATION.}\end{array}\right\}$ *.
nnnn	W	CB04008		FILE-LIMITS CLAUSE IS INVALID WITH * $\left\{\begin{array}{l}\text{MAGNETIC TAPE DEVICE.}\\\text{UNIT RECORD DEVICE.}\\\text{INDEXED ORGANIZATION.}\\\text{SEQUENTIAL ORGANIZATION.}\end{array}\right\}$ *.
nnnn	F	CB04009		ACTUAL KEY CLAUSE IS REQUIRED WITH RANDOM ACCESS.
nnnn	W	CB04010		ACTUAL KEY CLAUSE IS INVALID WITH * $\left\{\begin{array}{l}\text{MAGNETIC TAPE DEVICE.}\\\text{UNIT RECORD DEVICE.}\\\text{SEQUENTIAL ORGANIZATION.}\end{array}\right\}$ *.

LINE NUMBER	ERROR TYPE	ERROR CODE	CLAUSE NAME	MESSAGE TEXT
nnnn	W	CB04011		FORWARD KEY CLAUSE IS INVALID WITH * $\left\{\begin{array}{l}\text{MAGNETIC TAPE DEVICE.}\\\text{UNIT RECORD DEVICE.}\\\text{SEQUENTIAL ORGANIZATION.}\\\text{RELATIVE ORGANIZATION.}\\\text{RANDOM ACCESS.}\end{array}\right\}$ *.
nnnn	W	CB04012		ALTERNATE AREAS ARE NOT AVAILABLE WITH * $\left\{\begin{array}{l}\text{INDEXED ORGANIZATION.}\\\text{RANDOM ACCESS.}\end{array}\right\}$ *.
nnnn	F	CB04013	RERUN	* $\left\{\begin{array}{l}\text{RELATIVE ORGANIZATION}\\\text{INDEXED ORGANIZATION}\\\text{UNIT RECORD DEVICE}\end{array}\right\}$ * ILLEGAL WHEN USED FOR THE RERUN FILE.
nnnn	F	CB04014		INDEX-BLOCK SIZE IS REQUIRED WITH INDEXED ORGANIZATION.
nnnn	W	CB04015		INDEX-BLOCK SIZE IS ILLEGAL WITH * $\left\{\begin{array}{l}\text{UNIT RECORD DEVICE.}\\\text{MAGNETIC TAPE DEVICE.}\\\text{RELATIVE ORGANIZATION.}\\\text{SEQUENTIAL ORGANIZATION.}\end{array}\right\}$ *.
nnnn	F	CB04021	BLANK	CLAUSE ILLEGAL WITH * $\left\{\begin{array}{l}\text{ALPHANUMERIC ITEM.}\\\text{ALPHABETIC ITEM.}\\\text{ALPHANUMERIC EDITED ITEM.}\\\text{COMP-3/PACKED USAGE.}\\\text{COMP/BINARY USAGE.}\\\text{INDEX USAGE.}\\\text{OCCURS DEPENDING ON.}\\\text{GROUP ITEM.}\end{array}\right\}$ *.
nnnn	F	CB04021	JUSTIFIED	CLAUSE ILLEGAL WITH * $\left\{\begin{array}{l}\text{NUMERIC ITEM.}\\\text{ALPHANUMERIC EDITED ITEM.}\\\text{NUMERIC EDITED ITEM.}\\\text{COMP-3/PACKED USAGE.}\\\text{COMP/BINARY USAGE.}\\\text{INDEX USAGE.}\\\text{OCCURS DEPENDING ON.}\\\text{GROUP ITEM.}\end{array}\right\}$ *.
nnnn	F	CB04021	SYNCHRO-NIZED	CLAUSE ILLEGAL WITH * $\left\{\begin{array}{l}\text{INDEX USAGE.}\\\text{GROUP ITEM.}\end{array}\right\}$ *.
nnnn	F	CB04021	VALUE	CLAUSE ILLEGAL WITH * $\left\{\begin{array}{l}\text{INDEX USAGE.}\\\text{OCCURS.}\\\text{REDEFINES.}\end{array}\right\}$ *.
nnnn	F	CB04021	REDEFINES	CLAUSE ILLEGAL WITH OCCURS DEPENDING ON.

LINE NUMBER	ERROR TYPE	ERROR CODE	CLAUSE NAME	MESSAGE TEXT
nnnn	F	CB04021	OCCURS	CLAUSE ILLEGAL WITH 01 OR 77 LEVEL.
nnnn	F	CB04021	PICTURE	CLAUSE ILLEGAL WITH * { INDEX USAGE. / GROUP ITEM. } *.
nnnn	F	CB04022	* { BLANK / JUSTIFIED / SYNCHRO-NIZED } *	CLAUSE ILLEGAL WHEN SUBORDINATE TO ITEM WITH VALUE.
nnnn	F	CB04022	VALUE	CLAUSE ILLEGAL WHEN SUBORDINATE TO ITEM WITH * { REDEFINES. / OCCURS. / VALUE. } *.
nnnn	F	CB04023	USAGE	COMP OR COMP-3 IS ILLEGAL WITH * { ALPHANUMERIC ITEM. / ALPHABETIC ITEM. / ALPHANUMERIC EDITED ITEM. / NUMERIC EDITED ITEM. } *.
nnnn	F	CB04024	USAGE	COMP, COMP-3, OR INDEX IS ILLEGAL WHEN SUBORDINATE TO ITEM WITH VALUE CLAUSE.
nnnn	F	CB04025	VALUE	NUMERIC LITERAL ILLEGAL WITH * { ALPHANUMERIC ITEM. / ALPHABETIC ITEM. / ALPHANUMERIC EDITED ITEM. / NUMERIC EDITED ITEM. / GROUP ITEM. } *.
nnnn	F	CB04026	VALUE	ALPHANUMERIC LITERAL ILLEGAL WITH * { NUMERIC ITEM. / COMP-3/PACKED USAGE. / COMP/BINARY USAGE. } *.
nnnn	F	CB04027		FIGURATIVE CONSTANT OF 'ZERO' IS ILLEGAL WITH * { ALPHABETIC ITEM. / ALPHANUMERIC EDITED ITEM. / NUMERIC EDITED ITEM. } *.
nnnn	F	CB04028		OCCURS DEPENDING ON ILLEGAL WHEN SUBORDINATE TO ITEM WITH * { REDEFINES / OCCURS } * CLAUSE.
nnnn	F	CB04029		PICTURE REQUIRED FOR ELEMENTARY ITEM.
nnnn	U	CB04030	REDEFINES	CLAUSE INVALID FOR 01 RECORD DESCRIPTION -- REDEFINES CLAUSE IGNORED.

LINE NUMBER	ERROR TYPE	ERROR CODE	CLAUSE NAME	MESSAGE TEXT
nnnn	F	CB04031	OCCURS	THIS ITEM WOULD REQUIRE OVER 3 LEVELS OF SUBSCRIPTING -- OCCURS CLAUSE DROPPED.
nnnn	F	CB04032	OCCURS	MORE THAN ONE DEPENDING ON IN THE SAME RECORD IS NOT ALLOWED -- IGNORED.
nnnn	F	CB04033	USAGE	ELEMENTARY ITEM USAGE INCONSISTENT WITH USAGE OF DOMINANT GROUP -- GROUP USAGE ASSUMED.
nnnn	F	CB04034	VALUE	CLAUSE CANNOT BE SPECIFIED IN A RECORD WHEN DEPENDING ON OPTION WAS SPECIFIED.
nnnn	U	CB04035	VALUE	CLAUSE INVALID FOR ITEMS IN FILE SECTION OR LINKAGE SECTION.
nnnn	W	CB04038	VALUE	FILE LABEL ID EXCEEDS 17 CHARACTERS -- FIRST 17 CHARACTERS USED.
nnnn	W	CB04039	VALUE	MODIFICATION-CODE LITERAL EXCEEDS 4 CHARACTERS -- FIRST 4 CHARACTERS USED.
nnnn	W	CB04040	VALUE	NUMERIC LITERAL FOR * $\left\{\begin{array}{l}\text{LABEL ID}\\\text{RETENTION-PERIOD}\\\text{MODIFICATION CODE}\end{array}\right\}$ * MUST BE POSITIVE INTEGER -- VALUE DROPPED.
nnnn	W	CB04041	VALUE	NUMERIC LITERAL FOR * $\left\{\begin{array}{l}\text{RETENTION-PERIOD}\\\text{MODIFICATION-CODE}\end{array}\right\}$ * EXCEEDS 4 DIGITS -- FIRST 4 DIGITS ARE USED.
nnnn	W	CB04042	VALUE	FIGURATIVE CONSTANT SPECIFIED FOR RETENTION-PERIOD IS ILLEGAL -- ONLY ZERO ALLOWED.
nnnn	U	CB04043		ILLEGAL LEVEL NUMBER HIERARCHY WITHIN RECORD DESCRIPTION.
nnnn	W	CB04044		LABEL VALUE SPECIFIED FOR FILE WITH LABEL OMITTED.
nnnn	F	CB04045		FIRST LEVEL NUMBER OF RECORD DESCRIPTION MUST BE 01.
nnnn	F	CB04046		FIRST LEVEL NUMBER FOLLOWING 77 MUST BE 01.
nnnn	F	CB04047		77 ILLEGAL IN FILE SECTION -- CHANGED TO 49.

LINE NUMBER	ERROR TYPE	ERROR CODE	CLAUSE NAME	MESSAGE TEXT
nnnn	F	CB04048		77 MUST PRECEDE RECORD -- CHANGED TO 49.
nnnn	W	CB04049	PICTURE	SIZE OF THIS BINARY ITEM IS CHANGED TO * $\begin{Bmatrix}4\\8\end{Bmatrix}$ * DIGITS.
nnnn	F	CB04050	PICTURE	SIZE OF THIS BINARY ITEM EXCEEDS 8 DIGITS -- CHANGED TO 8.
nnnn	W	CB04051	PICTURE	BINARY ITEM IS NOT A SIGNED INTEGER.
nnnn	W	CB04052		LITERAL VALUE * $\begin{Bmatrix}\text{LEFT}\\\text{RIGHT}\end{Bmatrix}$ * TRUNCATED TO PICTURE SIZE -- SIGNIFICANT DIGITS ARE LOST.
nnnn	W	CB04053		ITEM IS UNSIGNED -- SIGN OF VALUE CLAUSE LITERAL IS DROPPED.
nnnn	F	CB04054		VARIABLE PORTION OF RECORD MUST BE AT END OF RECORD.
nnnn	F	CB05001		DUPLICATE SYSTEM ID'S IN SELECT SENTENCE -- SENTENCE IGNORED.
nnnn	F	CB05002		SYSTEM ID CANNOT BE SPECIFIED AS A FILE AND AS A RERUN DEVICE -- IGNORED.
nnnn	F	CB05003		FILE SPECIFIED IN SAME AREA CLAUSE MORE THAN ONCE -- LAST SPECIFICATION USED.
nnnn	F	CB05004		NUMBER OF FILES EXCEEDS THE MAXIMUM OF 16.
nnnn	F	CB05005		NO FILE DESCRIPTION TO MATCH SELECT ENTRY.
nnnn	F	CB05006		INVALID ACTUAL KEY -- KEY DROPPED.
nnnn	F	CB05007		INVALID FORWARD KEY -- KEY DROPPED.
nnnn	F	CB05008		DATA-NAME CANNOT BE SUBSCRIPTED -- SUBSCRIPT DROPPED.
nnnn	F	CB05009		SUBSCRIPT ERROR -- SUBSCRIPT DROPPED.
nnnn	F	CB05010		ACTUAL KEY MISSING.
nnnn	F	CB05011		SUBSCRIPT CANNOT BE AN OCCURRING ITEM -- SUBSCRIPT DROPPED.
nnnn	F	CB05012		FORWARD KEY MUST BE PRESENT WITH A START VERB.

LINE NUMBER	ERROR TYPE	ERROR CODE	CLAUSE NAME	MESSAGE TEXT
nnnn	F	CB05013		DATA-NAME IS NOT A RECORD AREA -- DATA-NAME DROPPED.
nnnn	F	CB05014	CLOSE	UNIT INVALID FOR RANDOM ACCESS MODE.
nnnn	F	CB05014	READ	AT END INVALID FOR RANDOM ACCESS MODE.
nnnn	F	CB05014	START	CLAUSE INVALID FOR RANDOM ACCESS MODE.
nnnn	F	CB05015	SEEK	CLAUSE REQUIRES ACCESS MODE RANDOM CLAUSE.
nnnn	F	CB05016	SEEK	CLAUSE INVALID FOR INDEX ORGANIZATION.
nnnn	F	CB05017	DELETE	CLAUSE INVALID FOR RELATIVE ORGANIZATION.
nnnn	F	CB05017	REWRITE	CLAUSE INVALID FOR RELATIVE ORGANIZATION.
nnnn	F	CB05017	START	CLAUSE INVALID FOR RELATIVE ORGANIZATION.
nnnn	F	CB05018	DELETE	CLAUSE INVALID FOR SEQUENTIAL ORGANIZATION.
nnnn	F	CB05018	OPEN	I/O INVALID FOR SEQUENTIAL ORGANIZATION.
nnnn	F	CB05018	READ	INVALID KEY INVALID FOR SEQUENTIAL ORGANIZATION.
nnnn	F	CB05018	SEEK	CLAUSE INVALID FOR SEQUENTIAL ORGANIZATION.
nnnn	F	CB05018	REWRITE	CLAUSE INVALID FOR SEQUENTIAL ORGANIZATION.
nnnn	F	CB05018	START	CLAUSE INVALID FOR SEQUENTIAL ORGANIZATION.
nnnn	F	CB05019	CLOSE	* $\left\{\begin{array}{l}\text{UNIT}\\ \text{NO REWIND}\\ \text{REEL}\end{array}\right\}$ * INVALID FOR UNIT RECORD FILES.
nnnn	F	CB05019	DELETE	CLAUSE INVALID FOR UNIT RECORD FILES.
nnnn	F	CB05019	OPEN	* $\left\{\begin{array}{l}\text{I/O}\\ \text{NO REWIND}\end{array}\right\}$ * INVALID FOR UNIT RECORD FILES.

LINE NUMBER	ERROR TYPE	ERROR CODE	CLAUSE NAME	MESSAGE TEXT
nnnn	F	CB05019	READ	INVALID KEY INVALID FOR UNIT RECORD FILES.
nnnn	F	CB05019	SEEK	CLAUSE INVALID FOR UNIT RECORD FILES.
nnnn	F	CB05019	WRITE	INVALID KEY INVALID FOR UNIT RECORD FILES.
nnnn	F	CB05019	REWRITE	CLAUSE INVALID FOR UNIT RECORD FILES.
nnnn	F	CB05019	START	CLAUSE INVALID FOR UNIT RECORD FILES.
nnnn	F	CB05020	CLOSE	UNIT INVALID FOR TAPE FILES.
nnnn	F	CB05020	DELETE	CLAUSE INVALID FOR TAPE FILES.
nnnn	F	CB05020	OPEN	I/O INVALID FOR TAPE FILES.
nnnn	F	CB05020	READ	INVALID KEY INVALID FOR TAPE FILES.
nnnn	F	CB05020	SEEK	CLAUSE INVALID FOR TAPE FILES.
nnnn	F	CB05020	WRITE	* $\left\{\begin{matrix}\text{INVALID KEY}\\ \text{BEFORE}\\ \text{AFTER}\end{matrix}\right\}$ * INVALID FOR TAPE FILES.
nnnn	F	CB05020	REWRITE	CLAUSE INVALID FOR TAPE FILES.
nnnn	F	CB05020	START	CLAUSE INVALID FOR TAPE FILES.
nnnn	F	CB05021	CLOSE	* $\left\{\begin{matrix}\text{NO REWIND}\\ \text{REEL}\end{matrix}\right\}$ * INVALID FOR MASS STORAGE FILES.
nnnn	F	CB05021	OPEN	NO REWIND INVALID FOR MASS STORAGE FILES.
nnnn	F	CB05021	WRITE	* $\left\{\begin{matrix}\text{BEFORE}\\ \text{AFTER}\end{matrix}\right\}$ * INVALID FOR MASS STORAGE FILES.
nnnn	F	CB05022		FIRST ELEMENTARY ITEM OF A RE-DEFINES MUST HAVE EVEN ADDRESS WHEN SYNCHRONIZED.
nnnn	U	CB05023		REDEFINES LENGTH IS NOT EQUAL TO THE REDEFINED LENGTH -- THE LARGER IS USED.
nnnn	W	CB05024		INTRA-RECORD SLACK BYTE INSERTED PRIOR TO THIS ITEM.

LINE NUMBER	ERROR TYPE	ERROR CODE	CLAUSE NAME	MESSAGE TEXT
nnnn	F	CB05025		RECORD SIZE EXCEEDS 16383 -- 16383 IS USED.
nnnn	W	CB05026		RECORD SIZE IS LESS THAN STATED MINIMUM -- COMPUTED SIZE USED.
nnnn	W	CB05027		RECORD SIZE IS GREATER THAN STATED MAXIMUM -- COMPUTED SIZE USED.
nnnn	F	CB05028		INVALID DATA-NAME SPECIFIED IN THE USING STATEMENT.
nnnn	F	CB05029		FORWARD KEY AND ACTUAL KEY ARE NOT EQUAL IN LENGTH -- ACTUAL KEY LENGTH USED.
nnnn	F	CB05030		INDEXED FILES CANNOT HAVE VARIABLE LENGTH RECORDS.
nnnn	W	CB05031		BLOCK SIZE MUST BE AN EVEN NUMBER OF BYTES -- ONE BYTE ADDED.
nnnn	F	CB05032		ACCESS MODE MUST BE SEQUENTIAL -- STATEMENT DROPPED.
nnnn	F	CB05033		A VALUE CLAUSE CANNOT BE SUB-ORDINATE TO ANOTHER VALUE CLAUSE -- SECOND DROPPED.
nnnn	F	CB05034		REDEFINED ITEMS MUST HAVE EQUAL LEVEL NUMBERS -- REDEFINES IGNORED.
nnnn	F	CB05035		REDEFINES AREA MUST IMMEDIATELY FOLLOW THE REDEFINED AREA -- REDEFINES IGNORED.
nnnn	U	CB05036		THE REDEFINED ITEM CANNOT BE SUBORDINATE TO OR HAVE AN OCCURS CLAUSE.
nnnn	U	CB05037		THE REDEFINED ITEM CANNOT BE SUBORDINATE TO OR HAVE A RE-DEFINES CLAUSE.
nnnn	F	CB05038		REDEFINED ITEM CANNOT HAVE AN OCCURS DEPENDING ON CLAUSE SUBORDINATE TO IT -- IGNORED.
nnnn	W	CB05039		LEFT TRUNCATION WILL OCCUR ON ALIT OR GROUP ITEM LITERAL.
nnnn	W	CB05040		RIGHT TRUNCATION WILL OCCUR ON ALIT OR GROUP ITEM LITERAL.
nnnn	F	CB05041		ID LABEL DATA-NAME INVALID -- DROPPED.

LINE NUMBER	ERROR TYPE	ERROR CODE	CLAUSE NAME	MESSAGE TEXT
nnnn	F	CB05042		RETENTION PERIOD LABEL DATA-NAME INVALID -- DROPPED.
nnnn	F	CB05043		MODIFICATION CODE DATA-NAME INVALID -- DROPPED.
nnnn	F	CB05044		THE BLOCK SIZE IS NOT LARGE ENOUGH TO FIT ONE RECORD.
nnnn	F	CB05045		BLOCKING FACTOR EXCEEDS 255 RECORDS.
nnnn	W	CB05046		INTER-RECORD SLACK BYTE INSERTED PRIOR TO THIS RECORD.
nnnn	W	CB05047		INTRA-RECORD SLACK BYTE INSERTED FOLLOWING THIS GROUP.
nnnn	F	CB05048		INVALID DATA NAME SUBSCRIPT. Subscript not defined as a numeric integer.
nnnn	F	CB05049		THIS LEVEL MUST BE SUBORDINATE TO THE ABOVE DEPENDING ON.
nnnn	F	CB05050		INVALID DEPENDING ON ID.
nnnn	F	CB05051		INVALID KEY MISSING.
nnnn	W	CB05052		ACTUAL KEY CANNOT EXCEED 8 DIGITS -- POSSIBLE TRUNCATION.
nnnn	F	CB05053		WORKING-STORAGE EXCEEDS 65KB.
nnnn	F	CB05054		RESIDENT LITERAL POOL EXCEEDS 65KB.
nnnn	F	CB05055		MORE THAN 255 77 AND 01 RECORD LEVELS IN LINKAGE SECTION.
nnnn	F	CB05056		RERUN SYSTEM ID REPLACED WITH SYSCHK.
nnnn	F	CB05057		BLOCK SIZE EXCEEDS 7294 CHARACTERS.
nnnn	F	CB05058		BLOCK SIZE LESS THAN 18 CHARACTERS.
nnnn	F	CB05059		DATA-NAME MUST BE SUBSCRIPTED -- SUBSCRIPT DROPPED.
nnnn	W	CB06001		EXIT PROGRAM OR STOP RUN NOT SPECIFIED.
nnnn	F	CB06002		SUBSCRIPT NOT IN RANGE OF TABLE.
nnnn	F	CB06003		INDEX DATA ITEM OR INDEX NAME REFERENCE ILLEGAL.
nnnn	F	CB06004		INDEX DATA ITEM ILLEGAL.

LINE NUMBER	ERROR TYPE	ERROR CODE	CLAUSE NAME	MESSAGE TEXT
nnnn	F	CB06005		DATA-NAME CANNOT BE JUSTIFIED.
nnnn	U	CB06006		SYSOUT CANNOT BE USED IN ACCEPT STATEMENT -- CONSOLE ASSUMED.
nnnn	U	CB06007		INTERMEDIATE RESULTS CANNOT EXCEED 18 DIGITS.
nnnn	F	CB06008		RECEIVING FIELD MUST NOT BE A LITERAL OR FIGURATIVE CONSTANT.
nnnn	W	CB06009		EXTERNAL NAME MUST NOT EXCEED 8 CHARACTERS -- TRUNCATED TO 8.
nnnn	W	CB06010		SYSIN CANNOT BE USED IN DISPLAY STATEMENT -- CONSOLE ASSUMED.
nnnn	F	CB06011		NUMERIC LITERAL MUST BE AN INTEGER IN EXAMINE STATEMENT.
nnnn	F	CB06012		NUMERIC LITERAL MUST BE UNSIGNED IN EXAMINE STATEMENT.
nnnn	W	CB06013		LITERAL MUST BE 1 CHARACTER IN EXAMINE STATEMENT.
nnnn	W	CB06014		POSSIBLE TRUNCATION -- IDENTIFIER MUST BE 3 CHARACTERS OR LESS.
nnnn	F	CB06015		ALPHABETIC TEST CANNOT BE USED WITH NUMERIC ITEM.
nnnn	F	CB06016		LITERALS OR INDEX NAMES ILLEGAL IN CLASS CONDITION.
nnnn	F	CB06017		NUMERIC LITERAL MUST BE UNSIGNED IN SET STATEMENT.
nnnn	F	CB06018		DATA-NAME MUST BE DISPLAY.
nnnn	F	CB06019		DATA-NAME MUST BE NUMERIC.
nnnn	F	CB06020		DATA-NAME MUST BE INTEGER.
nnnn	F	CB06021		MAXIMUM NUMBER OF OPERANDS EXCEEDED FOR STATEMENT.
nnnn	F	CB06022		NESTED CONDITIONALS NOT ALLOWED IN THIS LEVEL COBOL.
nnnn	F	CB06023		CLASS OF LITERAL INCONSISTENT WITH THAT OF IDENTIFIER.
nnnn	F	CB06024		ILLEGAL COMPARISON IN CONDITIONAL STATEMENT. Refer to the description of permissible comparisons in the **MRX/OS COBOL Reference** manual.

LINE NUMBER	ERROR TYPE	ERROR CODE	CLAUSE NAME	MESSAGE TEXT
nnnn	F	CB06025		NUMERIC ITEM MUST BE INTEGER.
nnnn	F	CB06026		ILLEGAL RECEIVING DATA ITEM IN MOVE STATEMENT. Refer to the description of permissible moves in the **MRX/OS COBOL Reference** manual.
nnnn	F	CB06027		RECEIVING DATA ITEM MUST BE INTEGER.
nnnn	F	CB06028		POSSIBLE TRUNCATION OF NUMERIC DATA ITEM OR NUMERIC LITERAL.
nnnn	F	CB06029		DATA-NAME CANNOT BE USED TO SET INDEX WHEN UP OR DOWN SPECIFIED.
nnnn	F	CB06030		PROCEDURE NAME IS MULTIPLY DEFINED.
nnnn	U	CB06031		NUMERIC LITERAL CANNOT BE ZERO IN SET STATEMENT.
nnnn	F	CB06032		NOT RELATION AND NOT CONDITION ILLEGAL IN CONDITIONAL STATEMENT.
nnnn	W	CB06033		NUMERIC TEST CANNOT BE USED WITH AN ALPHABETIC ITEM.
nnnn	F	CB06034		ILLEGAL COMBINATION IN SET STATEMENT.
nnnn	F	CB06035		POSSIBLE TRUNCATION -- DATA LENGTH MUST BE 2 DIGITS OR LESS.
nnnn	W	CB06036		EXIT STATEMENT MUST BE ONLY STATEMENT IN PARAGRAPH -- PERTAINS TO PREVIOUS PARAGRAPH.
nnnn	F	CB06037		LITERAL POOL ALLOCATION EXCEEDS 65KB FOR THIS SEGMENT.
nnnn	F	CB06038		REFERENCE TO UNDEFINED PROCEDURE NAME.
nnnn	F	CB06039		DATA-NAME MUST BE NUMERIC -- BLANK WHEN ZERO ILLEGAL.
nnnn	F	CB06040		DISPLAY BUFFER ALLOCATION EXCEEDS 65KB.
nnnn	W	CB07001		MESSAGE IS NOT AVAILABLE FOR THIS PASS OR PHASE.
nnnn	W	CB07002		POSSIBLE TRUNCATION.
nnnn	W	CB07003		NUGATORY ROUNDING.

LINE NUMBER	ERROR TYPE	ERROR CODE	CLAUSE NAME	MESSAGE TEXT
nnnn	W	CB07004		NUGATORY SIZE ERROR.
nnnn		CB07005		((RESERVE FOR FUTURE MESSAGE)).
nnnn		CB07006		((RESERVE FOR FUTURE MESSAGE)).
nnnn		CB07007		((RESERVE FOR FUTURE MESSAGE)).
nnnn	W	CB07008		ALTERABLE PERFORM EXIT -- GO TO GENERATED. An alterable paragraph (contains only a GO TO statement) is the exit paragraph of a PERFORM statement. This will result in the generation of a GO TO instead of a PERFORM.
nnnn	F	CB07009		ALTER REFERENCE TO A SECTION -- STATEMENT DROPPED.
nnnn	F	CB07010		REFERENCE TO AN UNALTERABLE PARAGRAPH -- STATEMENT DROPPED.
nnnn	F	CB07011		ALTER STATEMENT USED ACROSS INDEPENDENT SEGMENTS -- STATEMENT DROPPED.
		CB07012		((RESERVE FOR FUTURE MESSAGE)).
nnnn	F	CB07013		MAXIMUM NUMBER OF EXTERNAL-NAMES EXCEEDED -- TOO MANY CALL STATEMENTS USED.
nnnn	F	CB07014		RESIDENT LITERAL POOL EXCEEDS 65KB.
nnnn	F	CB07015		ILLEGAL USE OF PERFORM STATEMENT ACROSS SEGMENTS -- PERFORM STATEMENT DROPPED.
nnnn	W	CB09001		NUMBER OF REFERENCES EXCEEDS 21,588 -- REMAINING REFERENCES IGNORED.

COBOL SYSOUT FILE ERROR MESSAGES

There are four COBOL error messages that can appear on the SYSOUT file after the Control Language Services statement in error. Each message is preceded by an 8-digit error code that has the following format:

PP	SS	EEE	T

where:

PP	is always CB, specifying the COBOL compiler as the source of the error.
SS	is always 00 specifying the root of the COBOL compiler as the source of the error.
EEE	is a 3-digit error number specifying the error number within the root.
T	is a 1-digit number specifying the type of error. COBOL compiler errors listed in this file are all type 8, which is fatal.

The message text follows the error code.

ERROR CODE	MESSAGE TEXT
CB000018	COMPILER ABORT. This message is printed in the SYSOUT file when CB0X003 message is issued. As with the CB0X003 error, you should contact a systems engineer when this error occurs.
CB000028	LIST FILE NOT DEFINED, ABORT. The ID=LIST keyword parameter specification is missing from the //DEF statement.
CB000038	INPUT BUFFER TOO LARGE, CANNOT COMPILE. Not enough memory space was available for the input buffer. Either increase the size of the partition or decrease the block size of the input file.
CB000048	IMEM NOT DEFINED, ABORT. The IMEM= keyword parameter specified on the //PAR statement cannot be found in the input library.

OBJECT-TIME ERROR MESSAGES

COBOL messages that appear in the SYSOUT file specify compiler errors that occur during execution. When any of the first three errors listed below are encountered, the compiler will print the message in the SYSOUT file and continue processing. When the last error is encountered, the compiler will print the message in the SYSOUT file and terminate the current program by returning control to the calling program. The calling program can be either another COBOL program or the resident operating system.

ERROR CODE	LINE NUMBER	ERROR TYPE	MESSAGE TEXT
CB0E0018	nnnnn	Object Error	DATA CHECK
CB0E0028	nnnnn	Object Error	SUBSCRIPT RANGE ERROR
CB0E0038	nnnnn	Object Error	PARAMETER LIST TOO SHORT
CB0E0048	nnnnn	Object Error	PROGRAM DROP-OFF

The COBOL compiler supplies the line number (nnnnn) when it prints the message in the SYSOUT file. The line number refers to the line in the source listing where the error occurred.

INDEX

ACCEPT statement 8-43
Access methods
  random access
    and the ACCESS MODE clause 6-8
    and the ACTUAL KEY clause 6-8,F-3,F-4
    and the READ statement 8-37
    of indexed files 6-8,F-3
    of relative files 6-8,F-3
  sequential access
    and the ACCESS MODE clause 6-8
    and the ACTUAL KEY clause 6-8,F-4
    and the READ statement 8-37
    and the START statement 8-36
    of indexed files 6-8,F-3
    of relative files 6-8,F-3
    of sequential files 6-8,F-2
ACCESS MODE clause 6-8
ACTUAL KEY clause 6-8
ADD statement 8-11
Alignment, of elementary items 7-31
Alphabetic
  characters, defined 3-3
  items, in PICTURE clause 7-21
Alphanumeric
  characters, defined 3-3
  edited items, in PICTURE clause 7-23
  items, in PICTURE clause 7-21
ALTER statement 8-20
ALTERNATE AREAS option of RESERVE clause 6-6
Area A, definition 4-1
Area B, definition 4-1
Arithmetic expression characters 3-5
Arithmetic statements
  ADD 8-11
  DIVIDE 8-16
  GIVING option 8-11
  MULTIPLY 8-14
  overlapping operands 8-11
  ROUNDED option 8-11
  SIZE ERROR option 8-11
  SUBTRACT 8-12
ASSIGN clause 6-5
Asterisk
  in PICTURE clause 7-19
  used for comments 4-4
AUTHOR paragraph 5-1
Binary items, in PICTURE clause 7-23
BINARY option of USAGE clause 7-34
Blank lines, description 4-3
BLANK WHEN ZERO clause 7-16
Blanks 3-3,3-4
BLOCK CONTAINS clause 7-5
Block size, for indexed files G-1

Cataloging
  of object program 1-2,1-3
  of source program 1-1,1-2
CALL statement 8-48
Character set
  alphabetic characters 3-3
  alphanumeric characters 3-3
  arithmetic expression characters 3-5
  computer characters 3-3
  editing characters 3-4
  list 3-2
  numeric characters 3-3
  punctuation characters 3-3
  relation condition characters 3-5
  special characters 3-4
  word characters 3-3
Character strings
  constant 3-7
  definition 3-5
  name 3-6
  NOTE in Procedure Division 3-9
  PICTURE in Data Division 3-9
  word 3-5
Checkpoint records 6-15
CLOSE statement 8-45
Comment lines, description 4-4
Comparisons (see Relation condition)
Compiler
  input 1-1
  options 1-1
  output 1-3
Compiler directing sentences 8-3
Checkpoint/Restart program 6-16,6-16a |

Compiler-directing statements
  ENTER 8-52
  NOTE 8-53
COMPUTATIONAL option of USAGE clause 7-34
COMPUTATIONAL-3 option of USAGE clause 7-34
Computer characters 3-3
Conditional statements 8-2
Conditions
  class condition 8-4
  NOT condition 8-4
  relation condition 8-5
  test conditions 8-4
Configuration section 6-1
Connective, definition 3-6
Constants
  Literals 3-7
  Figurative constants 3-8
Continuation
  of lines 4-1
  of nonnumeric literals 4-1
  of words and numeric literals 4-1
Control characters D-1
CR symbol, in PICTURE clause 7-19
Cross reference list
  description 1-4
  suppression of 1-3
CURRENCY SIGN IS clause 6-3
Currency symbol, in PICTURE clause 7-19

Data description
  data description entries, definition 7-8
  data description entry clauses 7-9
  data item description entries, definition 7-9
  record description entries, definition 7-9
  maximum record or data item description entries 7-38
Data division 7-1
Data item
  description 7-8
  format 7-34
Data manipulation statements
  EXAMINE 8-33
  MOVE 8-29
Data map
  description 1-4
  suppression of 1-3
Data-name clause 7-10
Data-names, definition 3-6
Data organization
  indexed files
    access methods 6-8,F-3
    and the ACTUAL KEY clause 6-8
    and the DELETE clause 8-42
    and the FORWARD KEY clause 6-13
    and the INDEX-BLOCK clause 6-13
    and the START clause 8-36
    and the REWRITE clause 8-41
    definition F-2
  relative files
    access methods F-3
    and the ACCESS MODE clause 6-8
    and the ACTUAL KEY clause 6-8
    and the FILE-LIMIT clause 6-7
    and the SEEK clause 8-36
    definition F-2
  sequential files
    access methods 6-8,F-2
    and the ACCESS MODE clause 6-8
    definition F-1
DATA RECORDS clause 7-6
DATE-WRITTEN paragraph 5-1
DB symbol, in PICTURE clause 7-19
Decimal items, description 7-21
Decimal point
  in character set 3-4
  in PICTURE clause 7-18
DECIMAL POINT IS COMMA clause 6-3
Definitions, in glossary A-1
DELETE statement 8-42
DISPLAY option of USAGE clause 7-34
DISPLAY statement 8-44
DIVIDE statement 8-16
Division header, definition 4-3

EBCDIC, table B-1
Editing characters
  in PICTURE clause 7-17
  list 3-4
Editing, in PICTURE clause
  fixed insertion 7-26

floating insertion 7-27
simple insertion 7-25
special insertion 7-26
zero suppression 7-28
replacement editing 7-28
Elementary item, description 7-24
ELSE statement (see IF statement)
ENTER statement 8-52
Environment division
description 6-1
maximum use 6-18
Error messages H-1
EXAMINE statement 8-33
EXIT PROGRAM statement 8-52
EXIT statement 8-28
External decimal items, in PICTURE clause 7-21

Figurative constants 3-8
FILE-CONTROL paragraph
clause restrictions 6-14
description 6-4
maximum paragraph 6-15
File description entry
definition 7-3
description 7-5
maximum use 7-8
FILE-LIMIT clause 6-7
File-names, definition 3-6
File organization (see Data organization)
File processing summary F-1
File section, description 7-3
FILLER clause 7-10
Fixed insertion editing 7-26
Floating insertion editing 7-27
FOR MULTIPLE REEL/UNIT option of ASSIGN clause 6-5
Format notation 2-1
FORWARD KEY clause 6-13

GIVING option 8-11
Glossary of COBOL terms A-1
GO TO statement 8-18
Group item, description 7-24

HIGH-VALUE figurative constants 3-9

Identification division, definition 5-1
Identifier, definition 3-6
IF statement 8-9
Imperative statements
arithmetic 8-10
data manipulation 8-29
definition 8-10
input/output 8-34
procedure branching 8-18
subprogram linkage 8-48
INDEX-BLOCK clause 6-13,G-1
INDEX option of USAGE clause 7-34
Indexed files (see Data organization)
Indexing, in tables 9-5
Index-names, definition 3-7
Input-output section
description 6-4
maximum use 6-17
Input-output statements
ACCEPT 8-43
CLOSE 8-45
DELETE 8-42
DISPLAY 8-44
OPEN 8-35
READ 8-37
REWRITE 8-41
SEEK 8-36
START 8-36
WRITE 8-39
INSTALLATION paragraph 5-1
I-O CONTROL
description 6-15
maximum paragraph 6-16a |
Item size 7-20
Internal decimal items, in PICTURE clause 7-23

JUSTIFIED clause 7-16

Keyword, definition 3-5

LABEL RECORDS clause 7-7
Labels
LABEL RECORDS clause 7-7
VALUE OF clause 7-7
Level
concept of, in Data Division 7-2
indicators 7-3
numbers 7-3
in data description entries 7-9
Linkage section 7-4
Listings
error messages H-1
source program listing 1-4

suppressing source listing 1-2
Literals
Nonnumeric literals 3-8
Numeric literals 3-7
LOW-VALUE figurative constant 3-9
MEMORY SIZE clause 6-2
Messages, error H-1
Mnemonic-names, definition 3-6
MOVE statement 8-29
MULTIPLY statement 8-14

Names, definitions
data-names 3-6
file-names 3-6
index-names 3-7
mnemonic-names 3-6
procedure-names 3-6
program-names 3-7
system-names 3-7
NO REWIND option of OPEN statement 8-35
NOTE statement 8-53
Numeric
characters 3-3
edited items, in PICTURE clause 7-24
items, in PICTURE clause
binary 7-23
external decimal 7-21
internal decimal 7-23

OBJECT-COMPUTER paragraph 6-2
Object program, location 1-2,1-3
OCCURS clause
in the Data Division 7-17,7-38
used in table handling 9-8
OPEN statement 8-35
Optional word, definition 3-5
Organization of source program 3-1
Overlapping operands 8-11

Packed decimal 7-34
PACKED option of USAGE clause 7-34
Paragraphs, definition 4-3
PARAMETER statement 1-1
PERFORM statement 8-21
PICTURE clause 7-17
Priority numbers, in SECTION statement 9-14
Procedure branching statements
ALTER 8-20
EXIT 8-28
GO TO 8-18
PERFORM 8-21
STOP 8-28
Procedure division 8-1
Procedure map
description 1-4
suppression of 1-3
Procedure-names, definition 3-6
PROCESSING MODE clause 6-8
Program flow, COBOL 1-5
PROGRAM-ID paragraph, definition 5-1
Program-names, definition 3-7
Program termination 8-51
Punctuation characters 3-3

Quotation marks
defined in character set 3-3,3-4
enclosing non-numeric literals 3-8
QUOTE figurative constant 3-9

Random access (see Access methods)
READ statement 8-37
Reading/writing files (see Input-output statements)
RECORD CONTAINS clause 7-6
Record description entries 7-4
Recording modes E-1
REDEFINES clause 7-10
Relation condition
characters 3-5
comparison of nonnumeric operands 8-6
comparison of numeric operands 8-6
comparisons of index names and/or index data items 8-7
permissible comparisons of subject and object operands 8-7
used in table-handling 9-11
Relative files (see Data organization)
REMARKS paragraph 5-1
Replacement editing 7-28
RERUN clause 6-15
RESERVE clause 6-6
Reserved words
definition 3-5
list C-1
Rewind (see OPEN statement)
REWRITE statement 8-41
ROUNDED option 8-11
Right margin (RMARG) specification 1-2

SAME AREA clause 6-16a
Section header, definition 4-3
SECURITY paragraph 5-1
SEEK statement 8-36
SEGMENT LIMIT clause 9-15
Segmentation
  and the ALTER statement 9-15
  and the PERFORM statement 9-15
  fixed portion 9-13
  independent segments 9-13
  organization of Procedure Division 9-13
  segment classification 9-14
  segmentation control 9-14
  SEGMENT LIMIT clause 9-15
  structure of program segments 9-14
  priority numbers 9-14
  restrictions on program flow 9-15
SELECT clause 6-5
Sentences, definition 8-3
Sequence numbers 4-1
Sequential access (see Access methods)
Sequential files (see Data organization)
SET statement
  used in table handling 9-12
Simple insertion editing 7-25
SIZE ERROR option 8-11
Slack bytes
  inter-record 7-34
  intra-record 7-32
Slash (stroke), used for comments 4-4
SOURCE-COMPUTER paragraph 6-2
Source program
  error messages H-1
  input to compiler 1-1
  library facility 9-16
  listing, description 1-4
  location 1-1,1-2
Spaces, in character set 3-3,3-4
Spacing
  on source listing 1-2
  SPACE figurative constant 3-9
Special characters 3-4
Special features
  segmentation 9-13
  source program library 9-16
  table handling 9-1
Special insertion editing 7-26
SPECIAL-NAMES paragraph 6-3
Special registers 3-9
START statement 8-36
Statements
  compiler directing 8-2
  conditional 8-2
  imperative 8-2
STOP statement 8-28
Structure of source program 3-1
Subprogram linkage statements
  CALL 8-48
  EXIT PROGRAM 8-52
Subscripting
  in tables 9-3
  subscript checking 1-3
SUBTRACT statement 8-12
Suppression and replacement editing 7-28
Symbols
  repetition of 7-19
  used in PICTURE clause 7-17
SYNCHRONIZED clause 7-31
System-names, definition 3-7

Table handling
  Data Division considerations
    OCCURS clause 9-8
    USAGE clause 9-10
  defining a table 9-1
  indexing 9-5
  Procedure Division considerations
    relation condition 9-11
    SET statement 9-12
  references to table items 9-3
  subscripting 9-3
TALLY register 3-10,8-33

USAGE clause
  description 7-34
  used in table handling 9-10
USING clause
  in a CALL statement 8-48
  in the Procedure Division header 8-1,8-49

VALUE clause 7-36
VALUE OF clause 7-7

Word
  connective 3-6
  definition 3-3
  keywords 3-5
  optional words 3-5
  reserved words
    definition 3-5
    list C-1

Working-storage section	7-4
WRITE statement	8-39
Writing/reading files (see Input-output statements)	
ZERO figurative constant	3-9
Zero suppression and replacement editing	7-28

COMMENTS FORM

MRX/OS COBOL Reference Manual (2202.002)

Please send us your comments, to help us produce better publications. Use the space below to qualify your responses to the following questions, if you wish, or to comment on other aspects of the publication. Please use specific page and paragraph/line references where appropriate. All comments become the property of the Memorex Corporation.

- Is the material:

	Yes	No
Easy to understand?	☐	☐
Conveniently organized?	☐	☐
Complete?	☐	☐
Well illustrated?	☐	☐
Accurate?	☐	☐
Suitable for its intended audience?	☐	☐
Adequately indexed?	☐	☐

- For what purpose did you use this publication? (reference, general interest, etc.) ____________________
- Please state your department's function: ____________________
- Please check specific criticism(s), give page number(s), and explain below:

  ☐ Clarification on page(s) ____________________

  ☐ Addition on page(s) ____________________

  ☐ Deletion on page(s) ____________________

  ☐ Error on page(s) ____________________

First Class

Permit No. 14831
Minneapolis,
Minnesota 55427

Business Reply Mail

No Postage Necessary if Mailed in the United States

Postage Will Be Paid By

Memorex Corporation

Midwest Operations – Publications
8941 Tenth Avenue North
Minneapolis, Minnesota 55427

Thank you for your information..........

Our goal is to provide better, more useful manuals, and your comments will help us to do so.

..........Memorex Publications

Publications Bulletin

2202.002-0001
3/19/73

Release 2 Update Package for:

MRX/OS COBOL Level 1
Reference Manual
2202.002

This bulletin advises of changes that have occurred to the **COBOL Reference Manual** since the **November 1972** edition was issued. New and replacement pages are provided where required.

Pages	**Action**
Front Cover	Replace
v and vi	Replace
1-1 thru 1-4	Replace
3-1 thru 3-6	Replace
6-3 and 6-4	Replace
6-9 and 6-10	Replace
6-15 and 6-16	Replace
6-16a	Add
6-17 and 6-18	Replace
8-5 and 8-6	Replace
8-27 and 8-28	Replace
8-37 thru 8-40	Replace
8-45 and 8-46	Replace
9-7 and 9-8	Replace
G-1 thru G-6	Replace
H-1 thru H-4	Replace
H-4a	Add
H-7 and H-8	Replace
H-19 thru H-22	Replace
Index-1 thru Index-6	Replace

Technical changes to text, tables, and figures are marked with a vertical bar in the outer margin.

Pages containing *non-technical* changes (page layout, spelling corrections) are indicated by a bar opposite the page number.

Please file this bulletin with the publication to retain a record of changes.

Sequence Number: M101

MEMOREX

Computer System Products