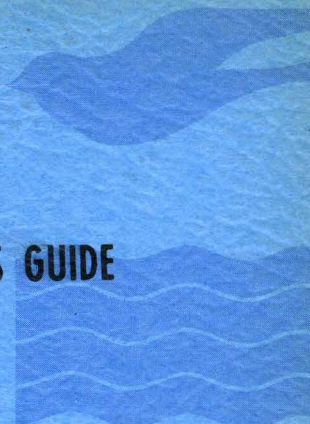


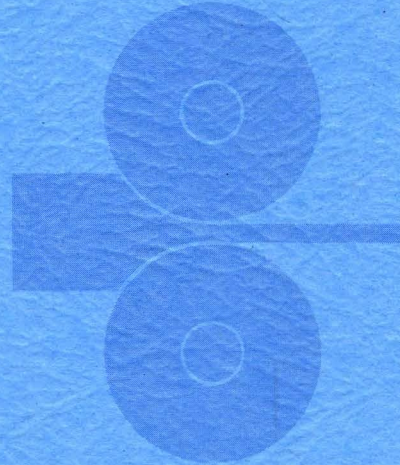
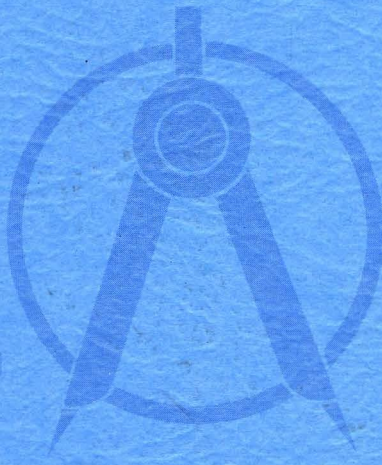
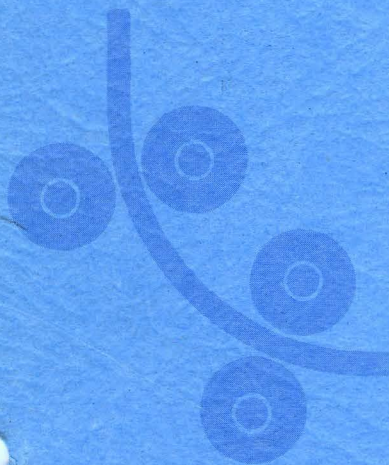
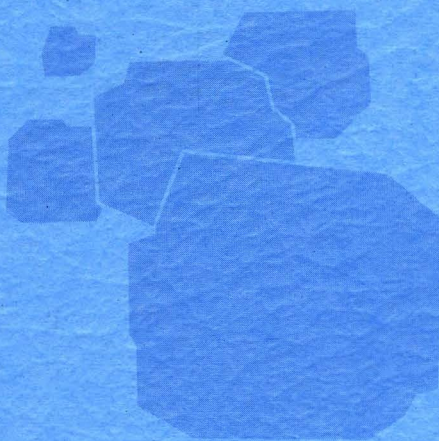


Honeywell



**COBOL
USER'S GUIDE**

**LEVEL 66
SOFTWARE**



USS Engineers and Consultants, Inc.
a Subsidiary of United States Steel Corporation

**SERIES 60 (LEVEL 66)/6000
COBOL USER'S GUIDE**

SUBJECT

Explanation of Language Elements, Coding Examples, Deck Setups, and Efficiency Techniques for Using Series 60 (Level 66) and Series 6000 COBOL

SPECIAL INSTRUCTIONS

This manual replaces *COBOL User's Guide*, Order No. BS09, for Series 6000 system users. Order No. BS09 must be used by Series 600 system users and by Series 6000 system users who are on prior software releases.

SOFTWARE SUPPORTED

**Series 60 (Level 66) Software Release 2
Series 6000 Software Release H**

ORDER NUMBER

DD26, Rev. 0

July 1975

Honeywell

PREFACE

The COBOL User's Guide is functionally organized into sections that provide information concerning COBOL concepts, Series 60/6000 implementation techniques, internal compiler characteristics, and efficiency considerations. In addition, sample deck setups and job control data are provided to assist the user in interfacing with the operating system.

Series 60 Level 66 is hereafter referred to as Series 60. The technical information contained in this manual refers to both the Series 6000 and Series 60 systems, unless otherwise specifically stated.

ACKNOWLEDGMENT

This acknowledgment has been reproduced from the "Journal of Development, 1968" as requested in that publication, prepared and published by the CODASYL COBOL Programming Language Committee.

"Any organization interested in reproducing the COBOL report and specifications in whole or in part, using ideas from this report as the basis for an instruction manual or for any other purpose is free to do so. However, all such organizations are requested to reproduce the following acknowledgment paragraphs in their entirety as part of the preface to any such publication. Any organization using a short passage from this document, such as in a book review, is requested to mention "COBOL" in acknowledgment of the source, but need not quote the acknowledgment.

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.

The authors and copyright holders of the copyrighted material used herein

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

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

FUNCTIONAL LISTING OF PUBLICATIONS
for
SERIES 60 (LEVEL 66) and SERIES 6000 SYSTEMS

FUNCTION	APPLICABLE REFERENCE MANUAL	ORDER NO.
<u>TITLE</u>		
<u>Series 60 (Level 66)/Series 6000:</u>		
Hardware reference:		
Series 60 Level 66 System	Series 60 Level 66 Summary Description	DC64
Series 6000 System	Series 6000 Summary Description	DA48
DATANET 355 Processor	DATANET 355 Systems Manual	BS03
DATANET 6600 Processor	DATANET 6600 Systems Manual	DC88
Operating system:		
Basic Operating System	General Comprehensive Operating Supervisor (GCOS)	DD19
Job Control Language	Control Cards Reference Manual	DD31
Table Definitions	System Tables	DD14
I/O Via MME GEINOS	I/O Programming	DB82
System initialization:		
System Startup	System Startup	DD33
System Operation	System Operation Techniques	DD50
Communications System	GRTS/355 and GRTS/6600 Startup Procedures	DD05
Communications System	NPS Startup	DD51
DSS180 Subsystem Startup	DSS180 Startup	DD34
Data management:		
File System	File Management Supervisor	DD45
Integrated Data Store (I-D-S)	I-D-S/I Programmer's Guide	DC52
Integrated Data Store (I-D-S)	I-D-S/I User's Guide	DC53
File Processing	Indexed Sequential Processor	DD38
File Input/Output	File and Record Control	DD07
File Input/Output	Unified File Access System (UFAS) (Series 60 only)	DC89
I-D-S Data Query System	I-D-S Data Query System Installation	DD47
I-D-S Data Query System	I-D-S Data Query System User's Guide	DD46
Program maintenance:		
Object Program	Source and Object Library Editor	DD06
System Editing	System Library Editor	DD30
Test system:		
Online Test Program	Total Online Test System (TOLTS)	DD39
Test Descriptions	Total Online Test System (TOLTS) Test Pages	DD49
Error Analysis and Logging	Honeywell Error Analysis and Logging System (HEALS)	DD44
Language processors:		
Macro Assembly Language	Macro Assembler Program	DD08
COBOL-68 Language	COBOL	DD25
COBOL-68 Usage	COBOL User's Guide	DD26
JOVIAL Language	JOVIAL	DD23
FORTRAN Language	FORTRAN	DD02
Generators:		
Sorting	Sort/Merge Program	DD09
Merging	Sort/Merge Program	DD09

FUNCTION

APPLICABLE REFERENCE MANUAL

ORDER
NO.TITLESeries 60 (Level 66)/Series 6000:

Simulators:

DATANET 355/6600 Simulation

DATANET 355/6600 Simulator

DD32

Service and utility routines:

Loader

General Loader

DD10

Utility Programs

Utility

DD12

Utility Programs

UTL2 Utility Routine (Series 60 only)

DC91

Media Conversion

Bulk Media Conversion

DD11

System Accounting

Summary Edit Program

DD24

FORTRAN

FORTRAN Subroutine Libraries

DD20

FNP Loader

DATANET 355/6600 Relocatable Loader

DD35

Service Routines

Service Routines

DD42

Software Debugging

Debug and Trace Routines

DD43

Time Sharing systems:

Operating System

TSS General Information

DD22

System Programming

TSS Terminal/Batch Interface

DD21

System Programming

TSS System Programmer's Reference

Manual

DD17

BASIC Language

Time Sharing BASIC

DD16

FORTRAN Language

FORTRAN

DD02

Text Editing

Time Sharing Text Editor

DD18

Remote communications:

DATANET 30/305/355/6600 FNP

Remote Terminal Supervisor (GRTS)

DD40

DATANET 355/6600 FNP

Network Processing Supervisor (NPS)

DD48

DATANET 700 RNP

RNP/FNP Interface

DB92

Transaction processing:

User's Procedures

Transaction Processing System User's
Guide

DD41

Handbooks:

System-operator communication

System Console Messages

DD13

Pocket guides:

Control Card Formats

Control Cards and Abort Codes

DD04

FORTRAN

FORTRAN Pocket Guide

DD82

CONTENTS

		Page
Section I	Introduction.	1-1
	General Description of COBOL	1-1
	COBOL User's Guide Organization.	1-2
	Language Features.	1-3
Section II	Representation of Data.	2-1
	Data Structures.	2-1
	Logical and Physical Records.	2-1
	Group Items and Elementary Items.	2-1
	Level-Numbers	2-2
	Noncontiguous Elementary Items.	2-3
	REDEFINES Entries	2-3
	Condition-Name Entries.	2-4
	RENAMES Entries	2-4
	Qualification	2-4
	Tables of Data Items.	2-6
	Subscripting	2-9
	Indexing	2-10
	External and Internal Data Formats	2-11
	External Data Formats	2-11
	Logical Record Format.	2-12
	Internal Data Formats	2-16
	Position Numbering	2-16
	The Machine Word	2-16
	Character-Strings.	2-17
	Binary Numbers	2-18
	Decimal Numbers.	2-21
	Data Description Entries	2-22
	DISPLAY Item Formats.	2-24
	COMPUTATIONAL Item Formats.	2-25
	Binary Representation of Fractional Values.	2-26
	COMPUTATIONAL Formats.	2-27
Section III	File Descriptions	3-1
	SELECT Sentence.	3-1
	OPTIONAL Phrase	3-1
	OVERLAY Phrase.	3-1
	File-Name Phrase.	3-2
	RENAMING Phrase	3-3
	ASSIGN Phrase	3-4
	FOR CARDS Phrase.	3-5
	FOR LISTING Phrase.	3-6
	FOR MULTIPLE REEL/UNIT Phrase	3-6
	RESERVE Phrase.	3-6
	Integer Option	3-6
	NO Option.	3-7
	FOR BLANK COMMON Phrase	3-7
	FILE-LIMIT(S) Phrase.	3-7
	ACCESS MODE Phrase.	3-7
	PROCESSING MODE Phrase.	3-8
	ACTUAL KEY Phrase	3-8
	APPLY Phrase	3-8.1

CONTENTS (cont)

	Page
PROCESS AREA Phrase	3-8.1
BLOCK SERIAL NUMBER Phrase.	3-9
SYSTEM STANDARD FORMAT Phrase	3-9
VLR FORMAT Phrase	3-10
RERUN Phrase.	3-10
SAME AREA Phrase.	3-10
SAME RECORD AREA Phrase	3-10
SAME SORT or SORT-MERGE AREA Phrase	3-11
MULTIPLE FILE Phrase.	3-11
File Description Entries	3-12
SORT-MERGE File Description Entries.	3-13
Level Indicator and File-Name	3-13
BLOCK CONTAINS Clause	3-13
DATA RECORD(S) Clause	3-16
LABEL RECORD(S) Clause.	3-17
OMITTED Option	3-17
STANDARD Option.	3-17
Label-Name Option.	3-21
RECORD CONTAINS Clause.	3-21.2
RECORDING MODE Clause	3-21.2
REPORT(S) Clause.	3-23
VALUE OF Clause	3-23
 Section IV	
Record Descriptions	4-1
Elementary Item Description Entries.	4-1
BLANK WHEN ZERO Clause.	4-1
Condition-Name Entry.	4-1
COPY Clause	4-2
JUSTIFIED Clause.	4-3
Level-Number/Data-Name Entries.	4-3
OCCURS Clause	4-4
PICTURE Clause.	4-8
Editing Rules.	4-10
REDEFINES Clause.	4-10
RENAMES Clause.	4-13
SYNCHRONIZED Clause	4-14
USAGE Clause.	4-17
DISPLAY Items.	4-18
COMPUTATIONAL Items.	4-18.1
INDEX Items.	4-18.3
VALUE Clause.	4-19
Groups of Elementary Items	4-20
 Section V	
File Processing	5-1
File Processing Concepts	5-1
File Declaration.	5-1
Sequential-Access Processing.	5-1
Random-Access Processing.	5-2
Open Status and Closed Status	5-2
Input, Output, and I-O Modes.	5-3
Special File Processing	5-3
Processing Optional Files.	5-3
Processing Nonlabeled Multiple Reel Files.	5-4
Processing Stranger Files via COBOL.	5-5
Relationship of Reporting Verbs to File Processing	5-7
Summary of File Property Relationships.	5-8
Assignment of Files.	5-10
File Control Cards.	5-10
System Standard Format.	5-10
Peripheral Devices.	5-12
Multiple File Tapes	5-12

CONTENTS (cont)

	Page
File Processing Areas	5-14
Buffer Areas	5-14
Record Areas	5-14
Sort Areas and Sort-Merge Areas	5-16
File Processing Statements	5-16
OPEN Statement	5-18
INPUT Option	5-19
I-O Option	5-19
NO REWIND Option	5-20
READ Statement	5-20
INTO Option	5-21
AT END Phrase	5-21
INVALID KEY Phrase	5-22
WRITE Statement	5-22
FROM Phrase	5-23
ADVANCING Phrase	5-23
INVALID KEY Phrase	5-25
SEEK Statement	5-25
CLOSE Statement	5-25
Standard Close File	5-26
Standard Close Reel	5-26
USE Statement	5-26
ERROR PROCEDURE Phrase	5-27
LABEL PROCEDURE Phrase	5-27
USE BEFORE REPORTING Phrase	5-28
File Processing Examples	5-29
Section VI	
Low-Volume Data Transmission	6-1
ACCEPT Statements	6-1
DISPLAY Statements	6-2
Data Transmission Techniques	6-3
System Input	6-3
System Output	6-3
Transaction Processing Interface	6-4
Remote Devices	6-4
Elapsed Processor Time	6-5
Date and Time	6-5
Console or Typewriter	6-6
Switches	6-7
Data Transmission Program Example	6-9
Section VII	
Transaction Processing System	7-1
Transaction Processing Executive (TPE)	7-1
TPAP Profile Table	7-1
Transaction Message Format	7-2
Transaction Processing Applications Programs (TPAPs)	7-3
TPE/TPAP Interface	7-3
Intercom Input File Processing	7-3
Input Subroutine (COMMI)	7-4
COBOL Input Statement Processing	7-5
Intercom Output File Processing	7-5
Output Subroutine (COMMO)	7-6
COBOL Output Statement Processing	7-7
Direct-Access (DAC) Mode Processing	7-8
Transaction Processing Applications Program Example	7-9
Section VIII	
Report Writer	8-1
Description of the Report Writer	8-1
Report Format	8-1
Report Control in the Procedure Division	8-2

CONTENTS (cont)

	Page
Skeletal Format for the Report Section	8-4
RD Entries	8-5
Report Group Entries	8-5
Elements of a Report	8-6
Report Groups	8-6
Control Data Items	8-7
Page Breaks and Overflow Breaks	8-8
File Characteristics	8-9
Line Counter	8-9
Page Counter	8-10
Report Writer Efficiency Techniques	8-10
SUM Counter Manipulation	8-10
Subtotalling	8-11
Rolling Forward	8-11
Crossfooting	8-12
SOURCE/SUM Correlation	8-13
Pre-Slew and Post-Slew Algorithms	8-14
Pre-Slew Calculations	8-14
Post-Slew Calculations	8-15
Combinations of Pre-Slew and Post-Slew Calculations	8-15
Report Writer Table Constraints	8-16
SUM Operand Limitations	8-16
RESET Stack Limitations	8-17
Report Table Capacity	8-19
Report Group Entries	8-20
Group Entries	8-21
SOURCE Entries	8-21
SUM Entries	8-23
VALUE Entries	8-24
Exceptions to Entry Sizes	8-25
Calculation of Report Group Size	8-25
Report Writer Program Example	8-27
Section IX	
File Ordering - SORT and MERGE	9-1
Concepts	9-1
Sorting	9-1
Merging	9-1
Ordering	9-1
Program Organization	9-2
SORT Statement	9-4
The Sort File	9-4
Sort Key Declarations	9-5
Variable-Length Records	9-6
Dominant Record Length	9-7
Sort Key Evaluation	9-7
Sort Equal Key Procedures	9-8
Engagement of Equal Key Procedures	9-8
Changing Equal Key Procedures	9-8
Disengagement of Equal Key Procedures	9-9
Sort Equal Key Record Processing	9-9
Sort Input Processing	9-9
USING Option	9-9
INPUT PROCEDURE Option	9-10
RELEASE Statement	9-10
Sort Output Processing	9-11
GIVING Option	9-11
OUTPUT PROCEDURE Option	9-11
RETURN Statement	9-12
Sort Operational Considerations	9-12
Flow of Control	9-12
Reserved File-Codes	9-14

CONTENTS (cont)

	Page
Implicit File Assignments	9-15
Input Files.	9-15
Output File.	9-15
Collation File-Codes	9-15
Borrowed File-Codes.	9-15
Collation File Manipulation	9-15
Sort Configuration.	9-17
Memory Assignment for Sort.	9-18
Dynamic Resource Allocation	9-19
Sort Examples	9-20
MERGE Statement.	9-22
The Merge File.	9-22
Merge Key Declarations.	9-22
Variable-Length Records.	9-22
Merge Key Evaluation.	9-23
Merge Equal Key Procedures.	9-23
Engagement of Equal Key Procedures	9-24
Changing Equal Key Procedures.	9-24
Disengagement of Equal Key procedures.	9-24
Merge Equal Key Record Processing.	9-25
Merge Input Processing.	9-25
Merge Output Processing	9-25
GIVING Option.	9-25
OUTPUT PROCEDURE Option.	9-26
RETURN Statement.	9-26
Merge Operational Considerations	9-27
Flow of Control	9-27
Reserved File-Codes	9-28
Implicit File Assignments	9-28
Input Files.	9-28
Output File.	9-28
Merge Configuration	9-28
Memory Assignment for Merge	9-28
Merge Examples.	9-29
SORT-MERGE Elective Options.	9-31
 Section X	
Data Movement Procedures.	10-1
MOVE Statement	10-1
Examples of MOVE Statements.	10-3
MOVE CORRESPONDING Statement	10-5
Examples of MOVE CORRESPONDING Statements.	10-7
REPLACING Phrase (EXAMINE Statement)	10-8
 Section XI	
Arithmetic Computations	11-1
Methods of Computation	11-1
Formulas	11-2
Unary Operators	11-3
Symbol Pairs.	11-3
Common Options in Statement Formats.	11-4
ROUNDED Option.	11-4
SIZE ERROR Option	11-5
CORRESPONDING Option.	11-5
Arithmetic Statements.	11-6
ADD Statement	11-7
COMPUTE Statement	11-7
DIVIDE Statement.	11-8
MULTIPLY Statement.	11-9
SUBTRACT Statement.	11-10
Multiple Results in Arithmetic Statements	11-10
Overlapping Operands.	11-11
Precision in Arithmetic Calculations.	11-11
TALLYING Phrase (EXAMINE Statement).	11-12

CONTENTS (cont)

	Page
Example of EXAMINE...TALLYING.	11-13
VARYING Phrase (PERFORM Statement)	11-14
Section XII	
Conditional Procedures.	12-1
Conditions	12-1
Simple Conditions	12-1
Relation Condition	12-1
Sign Condition	12-3
Class Condition.	12-3
Condition-Name Condition	12-4
Switch-Status Condition.	12-5
Compound Conditions	12-5
Abbreviated Combined Relation Conditions.	12-8
Abbreviation 1	12-8
Abbreviation 2	12-9
Use of the NOT Operator	12-11
Evaluation Rules for Conditions	12-14
Section XIII	
Table Handling.	13-1
Description of Table Handling.	13-1
Subscripting	13-2
Indexing	13-3
Rules for Subscripting and Indexing	13-4
Subscripting and Indexing - Sample Problem.	13-4
Size Restriction Upon Character-Oriented Arrays	13-6
SEARCH Statement	13-7
SET Statement.	13-9
SET Statement Rules	13-9
Comparisons Involving Index-Names and/or Index Data Items	13-10
SEARCH and SET Statement Examples	13-10
Section XIV	
Library Facility.	14-1
Description of the Library Facility.	14-1
COPY Functions	14-1
HIS COPY	14-1
HIS COPY Source Library Format.	14-2
Reference Listing Format.	14-4
Missing Library-Name.	14-4
Compressed Deck Options	14-4
HIS COPY WITH COMDK.	14-5
HIS COPY WITH CCOMDK	14-5
American National Standard COPY.	14-10
Library Format for American National Standard COPY.	14-10
Reference Listing Format.	14-11
Missing Library-Name.	14-11
Compressed Deck Options	14-12
American National Standard COPY WITH COMDK.	14-12
American National Standard COPY WITH CCOMDK	14-12
Section XV	
Segmentation and Modularization	15-1
Terminology.	15-1
Description of Segmentation.	15-1
Organization.	15-1
Segments.	15-2
Segment Classification.	15-2
Segmentation Control.	15-3
Structure of Program Segments	15-3

	Page
Priority-Numbers	15-3
Segment-Limit.	15-4
Transfer of Control	15-5
Restrictions on Transfer of Control and Program Alteration	15-5
Segmentation, Linking, and Loading.	15-6
Effects of Segmentation on Listings	15-10
Summary of Segmentation Requirements.	15-11
Description of Modularization.	15-11
Modules	15-12
Sections.	15-12
Procedure Division Communications	15-13
CALL Statement	15-14
ENTRY POINT Phrase	15-15
EXIT Statement	15-15
Data Compatibility.	15-16
File Compatibility.	15-16
Linked Overlay Environment Constraints	15-17
File Processing	15-17
ACCEPT and DISPLAY Statements	15-17
Overlay Management and Memory Organization.	15-17.1
Multiple Module Program Example	15-17.2
 Section XVI	
Efficiency Techniques	16-1
Optional DEBUG Statements.	16-1
Compilation Techniques	16-2
Unified Data Tables	16-3
Data-Name/FILLER Items	16-4
Group Items.	16-5
Elementary Items	16-5
Procedure Division Entries.	16-5
Resource Allocation	16-5
File Utilization.	16-7
Compilation Aborts.	16-8
Time-Saving and Space-Saving Techniques.	16-9
Input-Output Techniques	16-9
Incremental Report Printing Techniques.	16-10
Data Manipulation Techniques.	16-12
Data Description Techniques	16-13
Use of Compressed Decks.	16-14
Cross-Reference Facility	16-15
Packed Decimal Efficiency Techniques	16-15
Sample Business Program.	16-18
 Section XVII	
Obsolete Language Elements.	17-1
Environment Division Elements.	17-1.1
6000 WITH EIS Phrase.	17-1.1
Data Division Elements	17-1.1
PREPARED Option	17-1.1
Constant Section	17-2
Noncontiguous Constant Storage.	17-2
Constant Records.	17-3
VALUE of Constants.	17-3
Condition-Names	17-3
Tables of Constants	17-3
Data Description Entries	17-4
CLASS Clause.	17-4
Editing Clauses	17-5
FILE CONTAINS Clause.	17-6
POINT LOCATION Clause	17-7
RANGE Clause.	17-7
SEQUENCED Clause.	17-8

CONTENTS (cont)

	Page
SIGNED Clause	17-8
SIZE Clause	17-9
USAGE COMP-3 PACKED SYNC Clause	17-10
Procedure Division Elements.	17-11
Conditional Statements.	17-11
THEN Separator.	17-11
Appendix A Order of COBOL Source Program	A-1
Appendix B COBOL Deck Setups	B-1
Appendix C Nonstandard Feature Flagging.	C-1
Appendix D Collating Sequences	D-1
Appendix E COBOL Abort Codes	E-1
Appendix F Reserved GMAP Location Symbols.	F-1
Appendix G Compiler Limitations.	G-1
Index	i-1

ILLUSTRATIONS

Figure 2-1 Ranges of Fixed-Point Numbers	2-19
Figure 2-2 Ranges of Floating-Point Numbers.	2-20
Figure 9-1 Sort Program Organization	9-3
Figure 9-2 Merge Program Organization.	9-3
Figure 14-1 Library for Figures 14-2 and 14-3	14-5
Figure 14-2 HIS COPY WITH COMDK Option.	14-6
Figure 14-3 HIS COPY WITH CCOMDK Option	14-8
Figure 14-4 Library for Figures 14-5 and 14-6	14-13
Figure 14-5 American National Standard COPY WITH COMDK Option	14-14
Figure 14-6 American National Standard COPY WITH CCOMDK Option.	14-16

SECTION I

INTRODUCTION

GENERAL DESCRIPTION OF COBOL

COBOL is a computer language used primarily for programming business data processing operations.

The COBOL language offers various advantages to the computer user:

- It provides a rapid method of implementing complex programs.
- It reduces the cost of converting programs from one computer system to another.
- It reduces the time required to train personnel.
- It standardizes documentation.

Many changes and modifications have been introduced into the language since its inception in 1960; these changes resulted from user experience with practical applications.

The Series 60/6000 compiler was originally designed to provide the features of COBOL-61 Extended and was subsequently updated to conform to the COBOL-65 language specifications. Since then, it has been extended to provide most of the features of COBOL-68 as specified in American National Standard COBOL revision X3.23-1968. Consequently, whenever the term 'standard' is used in this document, it refers to American National Standard revision X3.23-1968. The compiler also provides some proprietary extensions as well as certain additional features specified in the CODASYL COBOL Journal of Development (JOD) and in the American National Standard COBOL revision X3.23-1974.

The Series 60/6000 COBOL Reference Manual, Order Number DD25, generally reflects the format of the Journal of Development and presents the formats, syntax rules, general rules, special considerations, and the strict interpretation of the language elements required to construct a COBOL source program. Unless otherwise indicated, the text of the reference manual conforms to the terminology expressed in the Journal of Development and/or the standard. In the reference manual, the formats and text describing extensions of the language are indicated by shading.

This COBOL User's Guide is a supplementary document to be used in conjunction with the reference manual; it is a compilation of COBOL concepts and programming techniques. It is organized into sections which are described below. The user's guide is not constrained to the exact wording of the standard or the Journal of Development although some of the text from these documents is used when it is meaningful in the discussions.

Certain language elements contained in previous versions of COBOL were deleted from the specifications prior to 1968 and are no longer included in the standard. These elements are no longer documented in the COBOL Reference Manual in order to discourage their continued use. For the benefit of users who have employed such features in the past, their implementation is retained in the compiler, with no guarantee that they will be maintained in the future. These obsolete elements are documented in Section XVII of this manual. It is recommended that they not be utilized in new programs and that they be eliminated from existing programs. To facilitate this, the compiler can optionally flag obsolete and/or nonstandard items on the COBOL source listing. The flagging feature is documented in Appendix C.

It is assumed that the reader of this user's guide has considerable knowledge of the contents of the COBOL Reference Manual. It is also suggested that the reader refer to the COBOL Pocket Guide as a quick reference source for formats and reserved words.

COBOL USER'S GUIDE ORGANIZATION

A brief summary of the contents of the sections contained in this manual is given below:

Section II provides an interpretation of data structures, internal and external data formats, and data descriptions.

Section III describes those language elements in the Environment and Data Divisions that apply to file descriptions.

Section IV describes record description entries and related usages.

Section V presents file processing concepts, some programming techniques, and the file processing statements.

Section VI describes low-volume data transmission using the ACCEPT and DISPLAY statements.

Section VII describes the Transaction Processing System feature using the ACCEPT MESSAGE and DISPLAY statements.

Section VIII describes the Report Writer feature, including information on report construction and table size restrictions.

Section IX is an overview of file ordering using the SORT and MERGE statements.

Section X presents examples of data manipulation using the MOVE and EXAMINE statements.

Section XI describes arithmetic computations and related examples.

Section XII describes the conditional and branching procedures.

Section XIII summarizes the Table Handling feature and presents the rules for subscripting and indexing.

Section XIV describes the COBOL Library Facility and presents related examples.

Section XV describes segmentation and modularization.

Section XVI examines some techniques designed to improve the efficiency of COBOL programs.

Section XVII contains a summary of language elements that have been deleted from COBOL.

Appendix A outlines the order of a COBOL source program.

Appendix B lists the \$ COBOL card system options and presents examples of COBOL deck setups.

Appendix C describes nonstandard feature flagging.

Appendix D lists the standard and commercial collating sequences.

Appendix E lists the COBOL abort codes.

Appendix F lists the reserved GMAP location symbols.

LANGUAGE FEATURES

COBOL allows computers to be programmed in a language that is similar to the English language. Paragraphs, sentences, and phrases are written, following the conventions of a standard reference format, to describe the data to be processed and to specify the required procedures. The resulting text is called a COBOL 'source program'.

The source program text consists of lines containing a maximum of 80 characters and is often keypunched on 80-column cards. The source program is submitted as input to the computer under the control of a special program known as a compiler. As output, the compiler produces an object program on punched cards, magnetic tape, or a mass storage device. The object program is the actual sequence of machine instructions required to accomplish the functions specified in the source program. In addition, the compiler produces an edited listing, which includes an annotated printout of the source program in the reference format. Another important function of the compiler is to analyze the source program for correct COBOL syntax, and to print error comments for any syntax errors that are detected. The computer's operation under control of the compiler is called compilation.

SECTION II

REPRESENTATION OF DATA

DATA STRUCTURES

Logical and Physical Records

A logical record, as described in the File Section or Working-Storage Section, is any set of contiguous data items considered as related due to content or usage. In an inventory transaction file, for example, each logical record could contain the information for a single transaction, or for all consecutive transactions pertaining to the same stock item, depending upon how the file is planned. The object program obtains input data from a peripheral file in units of one logical record each, and it prepares data for output in units of one logical record.

It is important to distinguish between the concepts of logical record and physical record. A physical record is the amount of data recorded by one physical write operation on a peripheral storage device, and may contain one or more logical records. In a COBOL source program, data and procedures are specified in terms of logical records. The compiler automatically supplies object program mechanisms to relate logical records to the physical records.

A logical record is normally subdivided into subordinate items, each of which is assigned a data-name. Each subordinate item may be further subdivided to permit more detailed references. Such items are referred to as group items and elementary items.

Group Items and Elementary Items

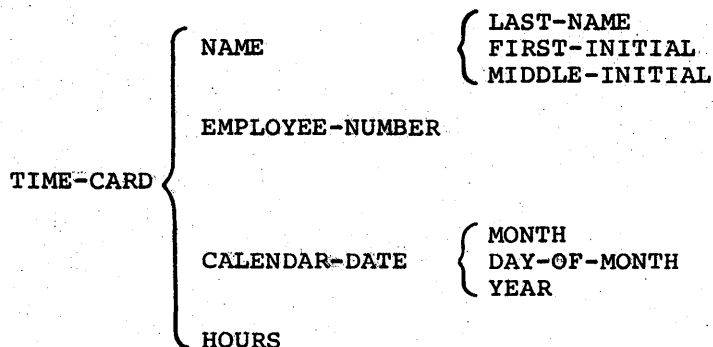
The term 'item', as used in COBOL, denotes either a group item or an elementary item. A logical record is usually a group item, since records are normally subdivided into subordinate items, but a logical record may be an elementary item.

Items that are themselves subdivided are called group items. In the following example, NAME, CALENDAR-DATE, and TIME-CARD are groups. A group consists of a sequence of subordinate groups and/or elementary items.

Elementary items are data items that are not additionally subdivided. In the following example, some of the elementary items are LAST-NAME, EMPLOYEE-NUMBER, DAY-OF-MONTH, and HOURS.

In the example, a weekly time card, the record is divided into four major items: NAME, EMPLOYEE-NUMBER, CALENDAR-DATE, and HOURS. If it is assumed that the group items CALENDAR-DATE and NAME are to be further subdivided, the record might be represented as follows:

Example:



Level-Numbers

In the Data Division, each record description entry begins with a level-number. A level-number is a one- or two-digit integer, whose value may range from 1 to 49, or may be 66, 77, or 88. Level-numbers less than 10 may be written with or without a leading zero (that is, 01 and 1 are equivalent). The level-numbers 66, 77, and 88 are reserved for special purposes and are described later.

Level-numbers show the organization of elementary items into groups and records. Records are the most inclusive groups possible, and are always assigned level-number 01 (or 1).

Less inclusive groups and elementary items within a record are assigned higher level-numbers, not greater than 49. Subordinate level-numbers need not be given successive numerical values.

Referring to the TIME-CARD example discussed previously, the hierarchical structure of the record can be described as follows:

- 01 TIME-CARD
 - 02 NAME
 - 03 LAST-NAME
 - 03 FIRST-INITIAL
 - 03 MIDDLE-INITIAL
 - 02 EMPLOYEE-NUMBER
 - 02 CALENDAR-DATE
 - 03 MONTH
 - 03 DAY-OF-MONTH
 - 03 YEAR
 - 02 HOURS

A group includes all groups and elementary items described after it until a level-number less than or equal to the level-number of that group is encountered. In the above example, MONTH, DAY-OF-MONTH, and YEAR make up a group called CALENDAR-DATE, because they are described immediately under it and have higher level-numbers. HOURS, however, is not a part of the group called CALENDAR-DATE, because its level-number is not greater than that of CALENDAR-DATE. The example shows that a group item (CALENDAR-DATE) and an elementary item (HOURS) may have the same level-number. It also shows that successive entries may be indented in a natural way, to make the hierarchical organization obvious.

An item may belong to more than one group. In the above example, the elementary item YEAR belongs to the group CALENDAR-DATE and also to the group TIME-CARD.

An entry immediately following the last elementary item of a group must have the same level-number as one of the groups to which the prior elementary item belongs. The following example is incorrect, because this rule restricts the level-number for EASY to be either 03 or 01:

```
01 ABLE
   03 BAKER
      04 CHARLIE
      04 DOG
02 EASY
```

Except in the Report Section, any level 01 item is considered a record, whether it is a group item or an elementary item, and its data-name is a record name.

Noncontiguous Elementary Items

Some elementary items in the Working-Storage Section have no relationship to one another and are not further subdivided. These items are called noncontiguous elementary items and are assigned the special level-number 77. When they are used, they must be the first entries in the section, preceding any record or group entries. Noncontiguous elementary items must not appear in the File Section or the Report Section.

REDEFINES Entries

A REDEFINES clause in a data description entry is used to apply a new description to the same memory area. The same memory area can be redefined as many times as necessary. Each REDEFINES clause describes a new data structure for the referenced area. The data items within that area may or may not correspond to any one of the described structures. For additional information on the use of the REDEFINES clause, refer to Section IV.

Condition-Name Entries

For some data items, certain values have significance. COBOL permits a condition-name to be used to determine that an item has a particular value or falls within a certain set of values. For example, the condition-name OUT-OF-STOCK might be associated with the value 0 for the item SUPPLY-ON-HAND in an inventory program. In the Procedure Division, the condition-name then provides a convenient and meaningful way to test the value of the item (that is: IF OUT-OF-STOCK...).

An elementary or group item in the File Section or Working-Storage Section may have condition-names assigned to any or all of its values. The condition-names are specified in entries immediately following the item to which they apply. Each condition-name entry is given level-number 88. An item with subordinate condition-name entries is called a conditional variable. A noncontiguous elementary item (with level-number 77) may be a conditional variable. Condition-name entries must not appear in the Report Section.

The following example illustrates the formation of condition-name entries:

```
03 GRADE PICTURE 99.  
   88 FIRST-GRADE VALUE IS 1.  
   88 SECOND-GRADE VALUE IS 2.  
   .  
   .  
   88 HI-SCHOOL VALUE 9 THRU 12.
```

RENAMES Entries

One or more RENAMES entries may be written as the last record description entry subordinate to a level 1 entry, for renaming or regrouping the items within the record. All RENAMES entries are assigned the special level-number 66. For additional information concerning the use of the RENAMES clause, refer to Section IV.

Qualification

The same data-name or condition-name may be assigned to two or more data items. The qualification feature of COBOL permits an item to be designated uniquely by appending the names of hierarchically more inclusive items as qualifiers to the data-name.

In any reference to a data item, its data-name may be qualified by the name of a group or file to which it belongs. The data-name is followed by either of the words IN or OF, and then by the group or file-name. The words IN and OF are considered synonymous. One or more such prepositional phrases may be required to uniquely identify the desired data item.

In any reference to a data item, although enough qualifiers must be written to make the data-name unique, it is not necessary to mention all of the data-names in a hierarchy unless they are needed to make the name unique. For a data-name which is unique in itself, no qualifiers are necessary.

A file-name is the highest level qualifier available. File-names must be unique in themselves; the same rule applies to report-names and noncontiguous elementary item names, and to record-names in the Working-Storage Section.

Assume that two records named MASTER and NEW-MASTER have the following hierarchical structures:

```
1 MASTER...
  2 CURRENT-DATE...
    3 MONTH...
    3 DAY-OF-MONTH...
    3 YEAR...
  2 LAST-TRANSACTION-DATE...
    3 MONTH...
    3 DAY-OF-MONTH...
    3 YEAR...
.
.
01 NEW-MASTER...
  02 CURRENT-DATE...
    03 MONTH...
    03 DAY-OF-MONTH...
    03 YEAR...
  02 LAST-TRANSACTION-DATE...
    03 MONTH...
    03 DAY-OF-MONTH...
    03 YEAR...
```

In the above example, two of the several data-names which must be qualified in all references are MONTH and DAY-OF-MONTH. The format of the qualified names would be:

MONTH { $\frac{\text{IN}}{\text{OF}}$ } CURRENT-DATE { $\frac{\text{IN}}{\text{OF}}$ } NEW-MASTER

DAY-OF-MONTH { $\frac{\text{IN}}{\text{OF}}$ } LAST-TRANSACTION-DATE { $\frac{\text{IN}}{\text{OF}}$ } MASTER

The specific rules for the use of qualification are:

- A qualifier must be the name of a group, record, or file that contains the item being qualified. Qualifiers must appear in ascending hierarchical order (that is, from elementary item-name up to record-name or file-name), and be separated by IN or OF.
- The same name must not appear at two levels in a hierarchy.
- If a data-name or condition-name is assigned to more than one data item in a source program, all references to the name require qualification, except where the COBOL rules specifically state that qualification is unnecessary.

- Any data-name requiring qualification must be qualified in every reference.
- A name may be qualified even if it is unique without qualification. Similarly, more qualifiers may be used than are actually needed for uniqueness. If more than one combination of qualifiers can ensure uniqueness, then any valid combination may be used.
- The data-name of a conditional variable may be used as a qualifier for any of its condition-names.
- Report data-names cannot be qualified by the file-name of the file to which the report is assigned.

Tables of Data Items

A table or array of data items is often required. The distinction between a table and an array is that an array is composed of elementary data items having identical data descriptions while a table may be composed of both elementary items and group items having differing data descriptions. Since the definition of array is a subset of the definition of table, the term 'table' will be used exclusively in the following discussion.

Successive item positions in a table may be numbered 1, 2, 3, 4, ...; therefore, any particular item can be identified by its position number. To refer to any particular item in the table, the data-name and the desired item's position number are given. The data-name would be ambiguous if a specific position number were not given.

A table described in a single entry is said to be 'one-dimensional'. COBOL permits the use of one-, two-, or three-dimensional tables, which are described, respectively, in one, two, or three data description entries. The number of occurrences in each dimension is specified via the OCCURS clause.

In a one-dimensional table, the number of occurrences of table items is specified in a single entry.

Example:

```
02 A; OCCURS...
```

In a two-dimensional table, the number of occurrences is specified in two entries; a group entry and a subordinate entry. The total number of occurrences of the elementary table items is the product of the numbers specified in the two entries.

Example:

```
02 MAJOR, OCCURS...
   03 MINOR; OCCURS...
```

In a three-dimensional table, the number of occurrences is specified in three entries; a group entry containing a subordinate group entry, which in turn contains another subordinate entry. The total number of occurrences of the elementary table items is the product of the numbers specified in the three entries which define the table.

Example:

```
02 MAJOR; OCCURS...
   03 INTERMEDIATE; OCCURS...
     04 MINOR; OCCURS...
```

The term 'dimensions' refers to the organization of the table. For example, suppose a two-dimensional table has three occurrences specified in the group entry and four occurrences in the elementary entry. The total table then has 12 items, of which the first four make up the first group, the second four make up the second group, etc. The tenth item in the table is actually the second item of the third group. The whole table resembles a page which is ruled into three horizontal rows and four vertical columns, and the tenth item appears in the third row, second column:

	Col 1	Col 2	Col 3	Col 4
Row 1	o	o	o	o
Row 2	o	o	o	o
Row 3	o	o	o	o

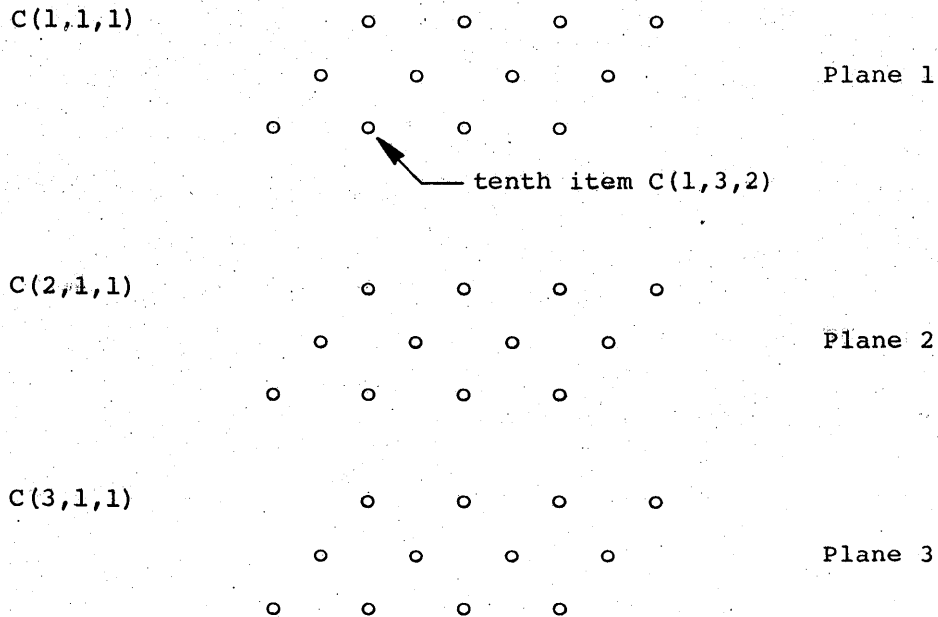
tenth item

The reference to any item is by its name, row number, and column number. In COBOL, such numbers are separated by a comma and enclosed in parentheses; that is, the tenth item position would be referred to by the data-name followed by (3,2). Such an expression is called a subscript.

Similarly, a three-dimensional table resembles a stack of ruled planes. The more inclusive group entry specifies the number of planes; the subordinate group specifies the number of rows in each plane, and the elementary entry specifies the number of columns in each row. Consider, for example, a table described as follows:

```
02 A OCCURS 3 TIMES.
   03 B OCCURS 3 TIMES.
     04 C OCCURS 4 TIMES.
```

The table described may be visualized as a stack of three ruled planes:



Note that item positions in plane 1 correspond exactly to those in the prior example, except that the plane number (1) must also be specified within the subscript. The tenth item of this table is C(1,3,2).

Consider a three-dimensional table which has two occurrences of the most inclusive group, five occurrences of the subordinate group, and 20 occurrences of the elementary item. Then:

1. The total table has 200 elementary items.
2. The first 20 items are the first row of the first plane.
3. The first 100 items make up the first plane.
4. The seventieth item in the table has position (1, 4, 10); that is, first plane, fourth row, tenth column.
5. References to successive table items after the seventieth item are made by incrementing the column number by unity up to (1, 4, 20). The next item is in a different row, so its position is (1, 5, 1).
6. The item following (1, 5, 20) has position (2, 1, 1).
7. The last table item has position (2, 5, 20).

COBOL permits table structures much more complicated than the examples described above, as long as no more than three dimensions are used. The minor (or only) OCCURS entry may be a group item:

```
02 A OCCURS 50 TIMES.  
03 B...  
03 C...  
.  
.
```

In the definition of a two- or three-dimensional table, other entries may intervene within the hierarchy of OCCURS entries:

```
02 D OCCURS 50 TIMES.  
03 E...  
03 F OCCURS 6 TIMES.  
04 G...  
04 H...  
04 I OCCURS 3 TIMES.  
05 J...  
05 K...  
04 L OCCURS 7 TIMES.  
03 M...  
03 N...  
04 P...  
04 Q...  
05 R OCCURS 2 TIMES.  
03 S OCCURS 12 TIMES.
```

Sometimes the number of significant items in a table varies throughout the execution of the object program. The variable number of occurrences is then specified via the DEPENDING option of the OCCURS clause. Such a table can only be a one-dimensional table.

SUBSCRIPTING

A subscript is a parenthesized expression whose value identifies the position of a particular table item. The subscript formats for references to table items depend upon the dimensions of the table, as follows:

```
One dimensional: (position-number)  
Two dimensional: (major, minor)  
Three dimensional: (major, intermediate, minor)
```

Subscripts must be enclosed in parentheses, as shown above, and commas or blanks must appear between the indicated items. In any reference to a table item, the parenthesized subscript must follow immediately after the terminal space of the table item's data-name. Multilevel subscripts are always written from left to right in the following order; major, intermediate, minor.

Column number, row number, and plane number must be positive integers or data-names. If data-names are used, they must specify items which will have positive integral values when the object program is executed. Integers are used when the desired table position is known in advance; data-names are used when the position depends upon data accessed or developed in the object program. A data-name within a subscript may not itself be subscripted, but it may be qualified if necessary for uniqueness.

A data item is said to be 'repeated' if the OCCURS clause is specified in the item's own description entry or in that of a group to which the item belongs. Any reference to a repeated item requires a subscript. The subscript must be one-, two-, or three-dimensional, reflecting the number of OCCURS entries affecting the desired item. Use of more than, or less than, the correct number of subscripts is illegal. A data-name can be subscripted only if the item is repeated. If a conditional variable is repeated, then references to its condition-names also require subscripts.

The first occurrence of an item or group is one, the second is two, etc. A subscript of (1,2) denotes the second item within the first group of the table. If a table consists of ten planes, each containing five rows, each containing three columns, the table is clearly three-dimensional, and its last item position is identified by subscript (10,5,3).

The following examples show some of the ways of referring to a particular item in a three-dimensional table of rates:

```
RATE (REGION, STATE, CITY)
RATE (3, STATE, CITY)
RATE (3, 5, 6)
```

In the third case, the actual table position of the item is computed during the compilation process. In other cases, the object program calculates the values, since the values of REGION, STATE, and CITY will become known only when the object program is executed.

The name of a repeated item may require qualification in references. If so, the entire subscript follows the last qualifier. Suppose group A occurs five times; within each occurrence, a subordinate group B occurs four times; and each group B contains an elementary item C with two occurrences. Several correct methods may be used to refer to the last table item, including the following:

```
C IN B IN A (5,4,2)
C IN B (5,4,2)
C IN A (5,4,2)
C (5,4,2)
```

(The last example assumes that no other item is named C.) The following references would violate the rule for combining subscripts and qualifiers:

```
C (5,4,2) IN B IN A
C (2) IN B (4) IN A (5)
C (4,2) IN A (5)
C (2) IN B (5,4)
```

INDEXING

Another method of specifying occurrence numbers is to affix one or more index-names to an item whose data description includes an OCCURS clause by using the optional INDEXED BY phrase. At object program execution, the contents of an index-name will correspond to an occurrence number for the specific dimension of the table with which the index-name is associated. Index-names must be initialized by using SET statements in the Procedure Division before being used as table references.

References are made to individual items in a table by specifying the name of the item followed by its related index-name in parentheses. The occurrence numbers required to complete the reference are obtained from the respective index-name; the index-name acting as a subscript. For references requiring more than one occurrence number, index-names and literals may be mixed but index-names and data-name subscripts may not be mixed. Thus, if indexing is to be used, each OCCURS clause within the hierarchy must contain an INDEXED BY phrase.

The value of an index-name can be modified only by using the PERFORM, SEARCH, and SET statements. If a data item is described with USAGE INDEX, the SET statement may be used to move data between the data item and an index-name. Data items described with USAGE INDEX are called index data items.

Table items can be accessed through direct or relative indexing. Direct indexing is using index-names in the same manner as subscripts are used. Relative indexing is the practice of inserting a space delimited operator (+ or -) and an integer following the index-name. The occurrence number to which the setting of the index-name corresponds is effectively incremented or decremented by the value of the integer. Relative indexing does not, however, alter the value of the index-name.

EXTERNAL AND INTERNAL DATA FORMATS

In COBOL, the representation of data within memory is called the 'internal' format, while the representation of data on peripheral devices is called the 'external' format.

External Data Formats

The external format of a file is the manner in which it is represented on a peripheral device.

Certain general considerations apply to the file:

1. The actual peripheral device used.
2. The MULTIPLE FILE option (applicable only to magnetic tape files).
3. Recording mode (which on magnetic tape may be binary or BCD, with high or low density).
4. Presence or absence of label records.

Other considerations pertain to the contents of each data block (physical record) of the file:

1. Presence or absence of a block serial number.
2. Blocking factor (number of logical records per block) or block size (number of data characters or computer words per block).

3. Logical record format; FLR (fixed-length records) or VLR (variable-length records).

A special external format is used for records in a file when OCCURS...DEPENDING has been specified in record description entries. Otherwise, the external format on disk or a magnetic tape recorded in binary mode is exactly the same as the internal format.

On BCD tape, the formats are the same except that the six-bit binary codes representing the data characters are generally different from the codes used in memory.

On printer listings, external and internal formats are equivalent. (Two characters, ? and !, are nonprinting characters reserved for printer carriage control.)

External and internal formats are equivalent for punched cards.

LOGICAL RECORD FORMAT

In Series 60/6000 COBOL, each logical record in a file begins in the first character position of a computer word. A logical record consisting of a single 21-character item would be stored as follows:

```
d d d d d d First word
d d d d d d
d d d d d d
d d d x x x Last word
```

In this example, the character positions represented by d contain data, while those represented by x are unused. Unused positions appear as shown both in memory and on the peripheral device.

Fixed-Length Records

The fixed-length record (FLR) format may be used for files whose logical records require the same number of computer words. In the FLR format, the uniform record size is established from the record description entry. In a block of several FLR records, the successive records are adjacent to each other, with no intervening control information. A FLR block composed of records similar to that shown in the preceding Logical Record Format paragraph would appear as follows:

```

d d d d d d }
d d d d d d } First record
d d d d d d }
d d d x x x }
d d d d d d }
d d d d d d } Second record
d d d d d d }
d d d x x x }

```

Variable-Length Records

Circumstances requiring the variable-length record (VLR) format are:

1. Two or more data records of unequal sizes (in computer words) have been described for the file. In this case, the size of each record type is fixed, but the record type may vary from one record to the next in the file.
2. The file is to receive one or more reports generated by the Report Writer.
3. The file is to reside on a mass storage device.
4. The file is to have system standard format.
5. The OCCURS...DEPENDING clause appears in the description of one or more of the data records of the file. The size of an OCCURS...DEPENDING table varies from one record to the next, causing the record size to vary, even if only one data record type has been specified for the file.

Each logical record in a VLR file is preceded by a record control word on the peripheral device and in the input-output buffers. The record control word is supplied and interpreted automatically by the input-output routines and is not accessible to the object program. The record control word occupies one computer word and has the following format:

0	17 18	23 24	29 30	35
record size in words (binary integer)	zero	media code	report code	

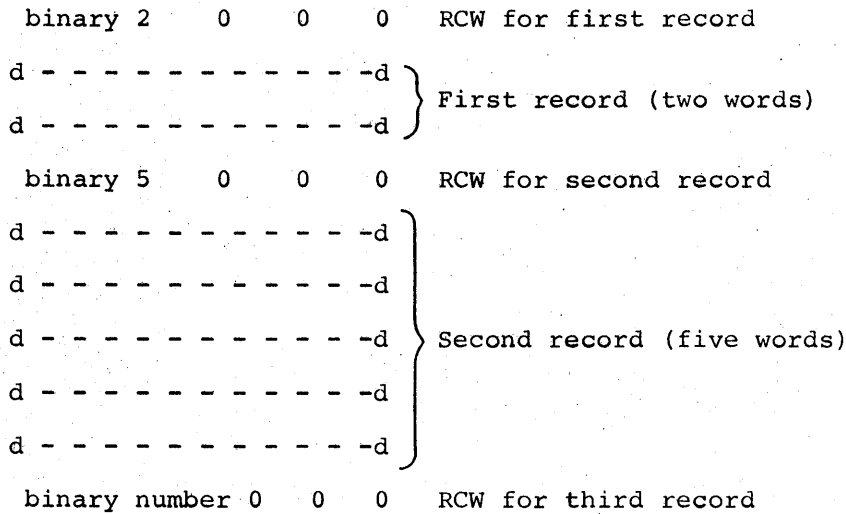
The record control word is not considered to be a part of the logical record and is not counted in the record size.

The media code is determined as follows:

- 2 - The record is intended to be treated as a card image even though it may appear on any peripheral device except a random-access mass storage device.
- 3 - Either this record is a report line generated by a Report Writer or it is intended to be treated as a printed line image even though it may appear on any peripheral device except a random-access mass storage device.
- 0 - The media code is zero except under the circumstances described above.

Except in certain Report Writer functions, the report code is zero, % for system output, or the last character of the two-character file-code for files intended to be treated as printed line images. The Report Writer applications of the report code are described in Section VIII and the remaining applications are described under the WRITE statement in Section V.

In a block of VLR records, the record control word intervenes between the successive records. A VLR block beginning with a 12-character record followed by a 30-character record would appear as follows:



The recording mode must be binary for any file utilizing VLR format.

Partitioned Records

A partitioned record is a logical record that is larger than the size of the physical record available to contain it. The input-output routines process such large records automatically by splitting them over 320-word blocks using a record control word for each block to control the splitting and reconstruction of large logical records.

To use this facility, the APPLY SYSTEM STANDARD FORMAT and APPLY PROCESS AREA phrases must be specified either explicitly or implicitly for the file. (See Section V.) Partitioned records are not required for random-access files because the logical records in these files are not limited by a block size of 320 words. The partitioning of records is not allowed for records that contain OCCURS...DEPENDING ON fields.

When a record is partitioned, each physical block of 320 words is termed a logical record segment and a logical record segment number is placed in each record control word except the first. A logical record segment code is placed in bits 24 and 25 of each record control word. Except for the logical record segment code and number fields, the rules governing record control words also apply to partitioned files.

The format of the first record control word for a partitioned record is:

0	18	23	25	26	30	35
segment size in words	zero	01	media	code	report	code

Intermediate logical record segments of a partitioned record have record control words with the following format:

0	18	23	25	26	35
segment size in words	zero	10	segment number		

The last logical record segment of a partitioned record has a control word with the following format:

0	18	23	25	26	35
segment size in words	zero	11	segment number		

Internal Data Formats

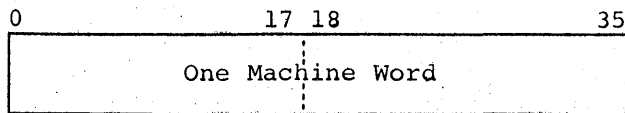
The processor is functionally organized to process 36-bit groupings of information. Special features are also included for ease in manipulating four-bit groups, six-bit groups, nine-bit groups, 18-bit groups, and 72-bit double-precision groups. These bit groupings are used by the hardware and software to represent a variety of forms of information.

POSITION NUMBERING

The numbering of bit positions, character positions, words, etc., increases in the direction of conventional reading and writing: from the most to the least significant digit of a number, and from left to right in conventional alphanumeric text.

THE MACHINE WORD

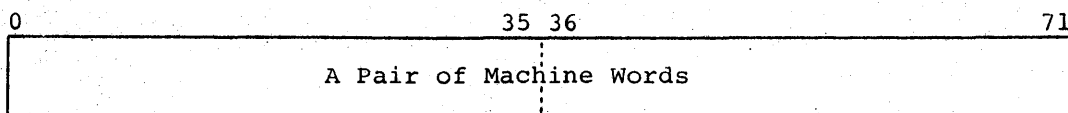
The machine word consists of 36 bits arranged as follows:



Upper Half-Word Lower Half-Word

Data transfers between the processor and memory are word oriented; 36 bits are transferred at a time for single-precision data and two successive 36-bit word transfers occur for double-precision data.

The processor has many built-in features for transferring and processing pairs of words. In transferring a pair of words to or from memory, a pair of memory locations is accessed; these addresses are an even number and the next higher odd number. A pair of machine words is arranged as follows:



Even Address

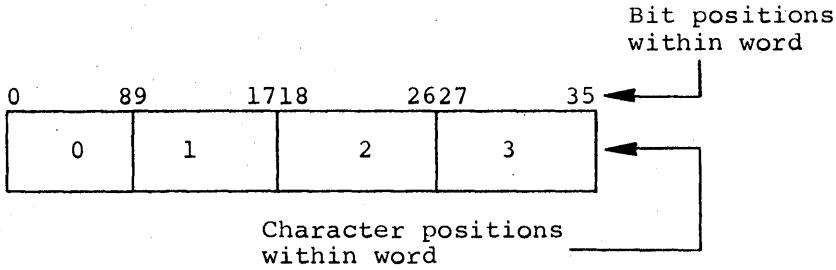
Odd Address

CHARACTER-STRINGS

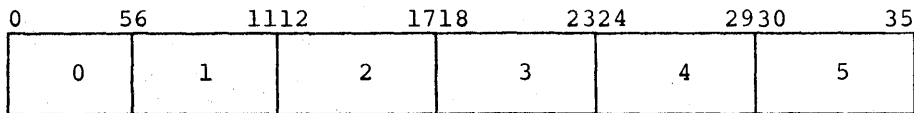
Character Positions

Alphanumeric data is represented by four-bit, six-bit, or nine-bit characters. A machine word contains either eight, six, or four characters, respectively. The character positions within the word are as follows:

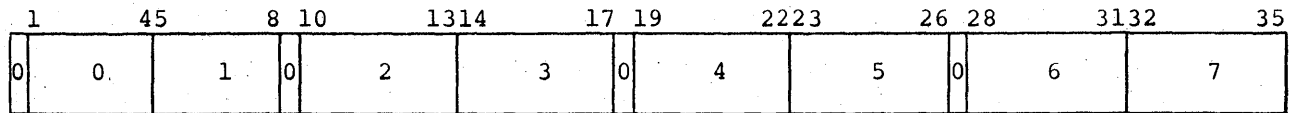
9-Bit Characters:



6-Bit Characters:

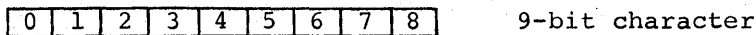


4-Bit Characters:



Bit Positions

Bit positions within a character are as follows:

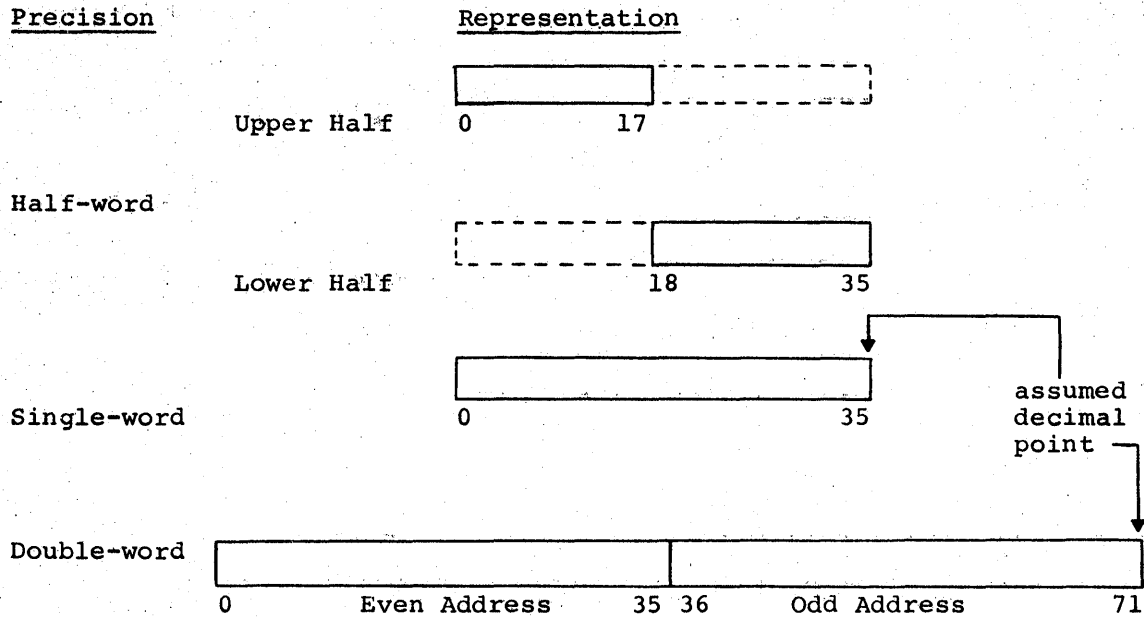


Thus, bit and character positions increase from left to right as in normal reading.

BINARY NUMBERS

Fixed-Point Numbers

Binary fixed-point numbers are represented with half-word, single-word, and double-word precision as shown below.



For algebraic operations, operands and results are regarded as signed binary numbers, and the leftmost bit is used as a sign bit (a 0 being plus and 1 minus). When the sign is positive, all the bits represent the absolute value of the number; when the sign is negative, they represent the two's complement of the absolute value of the number.

In the case of addition and subtraction, the occurrence of an overflow is reflected by the carries into and out of the leftmost bit position (the sign position). If the carry into the leftmost bit position does not equal the carry out of that position, then overflow has occurred. If overflow has been detected and if the sign bit equals 0, the resultant is below range; if with overflow the sign bit equals 1, the resultant is above range.

In integer arithmetic, the location of the decimal point is assumed to the right of the least significant bit position; that is, depending on the precision, to the right of bit position 35 or 71.

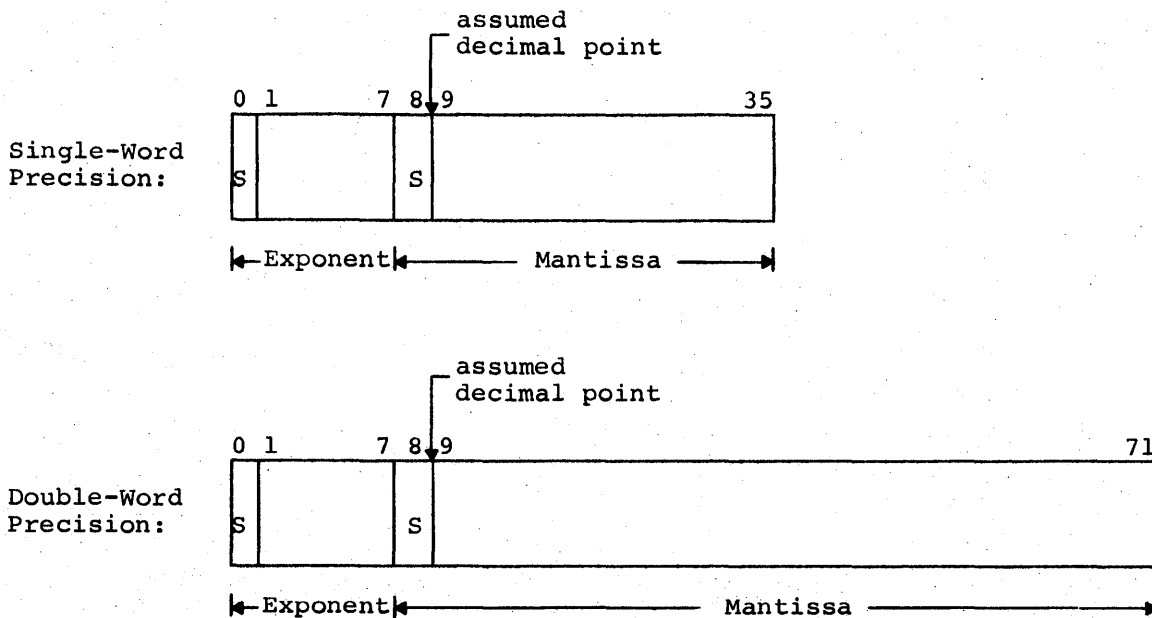
The number ranges for the various cases of precision, interpretation, and arithmetic are given in Figure 2-1.

Inter-pretation	Arithmetic	Precision		
		Half-Word (Xn, Y _{0...17})	Single-Word (A,Q,Y)	Double-Word (AQ, Y-pair)
Algebraic	Integral	$-2^{17} \leq N \leq (2^{17}-1)$	$-2^{35} \leq N \leq (2^{35}-1)$	$-2^{71} \leq N \leq (2^{71}-1)$
	Fractional	$-1 \leq N \leq (1-2^{-17})$	$-1 \leq N \leq (1-2^{-35})$	$-1 \leq N \leq (1-2^{-71})$
Logic	Integral	$0 \leq N \leq (2^{18}-1)$	$0 \leq N \leq (2^{36}-1)$	$0 \leq N \leq (2^{72}-1)$
	Fractional	$0 \leq N \leq (1-2^{-18})$	$0 \leq N \leq (1-2^{-36})$	$0 \leq N \leq (1-2^{-72})$

Figure 2-1. Ranges of Fixed-Point Numbers

Floating-Point Numbers

Binary floating-point numbers are represented with single-word and double-word precision. The upper eight bits represent the integral exponent in two's complement form, and the lower 28 or 64 bits represent the fractional mantissa in two's complement form. The format for a floating-point number is:



where S = sign bit

Before performing floating-point additions or subtractions, the processor aligns the number that has the smaller positive exponent. To maintain accuracy, the lowest permissible exponent of -128 together with the mantissa equal to 0.00....0 has been defined as the machine representation of the number zero (which has no unique floating-point representation). Whenever a floating-point operation yields a resultant untruncated machine mantissa equal to zero (71 bits plus sign because of extended precision), the exponent is automatically set to -128.

Normalized Floating-Point Numbers

For normalized floating-point numbers, the binary point is placed at the left of the most significant bit of the mantissa (to the right of the sign bit). Numbers are normalized by shifting the mantissa (and correspondingly adjusting the exponent) until no leading zeros are present in the mantissa for positive numbers, or until no leading ones are present in the mantissa for negative numbers. Zeros fill in the vacated bit positions.

The number ranges resulting from the various cases of precision, normalization, and sign are given in Figure 2-2.

	Sign	Single Precision	Double Precision
Normalized	Positive	$2^{-129} \leq N \leq (1-2^{-27})2^{127}$	$2^{-129} \leq N \leq (1-2^{-63})2^{127}$
	Negative	$-(1+2^{-26})2^{-129} \geq N \geq -2^{127}$	$-(1+2^{-62})2^{-129} \geq N \geq -2^{127}$
Unnormalized	Positive	$2^{-155} \leq N \leq (1-2^{-27})2^{127}$	$2^{-191} \leq N \leq (1-2^{-63})2^{127}$
	Negative	$-2^{-155} \geq N \geq -2^{127}$	$-2^{-191} \geq N \geq -2^{127}$

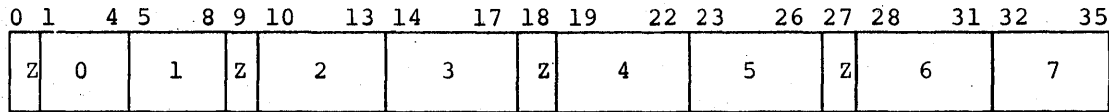
NOTE: The floating-point number zero is not included in the figure.

Figure 2-2. Ranges of Floating-Point Numbers

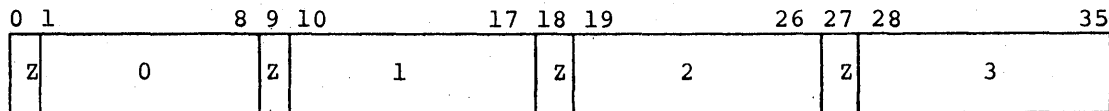
DECIMAL NUMBERS

Scaled decimal numbers are expressed as decimal digits in either the four-bit or nine-bit character formats. They are expressed as unsigned numbers or as signed numbers using a separate sign character.

Decimal data utilizes the following formats:



Packed Decimal (4-bit)



ASCII (9-bit)

The 'Z' in the bit positions represents the bit value 0 while other numbers in the fields represent the character positions.

Decimal Data Character Codes

During arithmetic operations, digits and signs are checked by the hardware as four-bit data (the four least significant bits from a nine-bit numeric). The following interpretations are made:

Bit Pattern for Character	Interpreted as	Abort if
0000	0	found where descriptor specifies sign
0001	1	
0010	2	
0011	3	
0100	4	
0101	5	
0110	6	
0111	7	
1000	8	
1001	9	
1010	+	found where descriptor specifies digits
1011	+	
1100	+	
1101	-	
1110	+	
1111	+	

Because of the above interpretations, the movement of SPACES to a group item containing decimal data will result in object program aborts when that data is referenced in arithmetic statements.

The following codes (nine-bit zones are created by prefixing binary 00010) are generated for output signs; the values are in octal:

	Plus	Minus
4-bit	14	15
9-bit	053	055

DATA DESCRIPTION ENTRIES

In COBOL, data items are described in terms of a standard data format. The description of each data item states its conceptual properties, rather than its representation within the computer. This conceptual description will imply specific physical representations in the computer.

Each data item is described as a string of characters. The basic properties of a data item are its size and its class. The size is the number of characters the item contains. The class may be alphabetic, numeric, or alphanumeric, depending upon the type of characters of which it is composed. In addition, the description may specify the placement of an assumed sign or decimal point, or how the data item should be edited for printing.

Every data item is described by a record description entry. The record description entry begins with a level-number and a data-name. (If no reference is to be made to the item, the reserved word FILLER may appear instead of a user-supplied data-name.) The functions of the level-number and the data-name are as follows:

- The level-number indicates any relationship this item may have with the items described in adjacent entries.
- The data-name provides a means whereby the item can be referred to elsewhere in the program.

The remainder of the entry consists of descriptive clauses chosen from the following list (in which the clauses appear in approximate order of importance):

- PICTURE - Gives the detailed format of the elementary item.
- VALUE - Specifies the initial value of the item (applicable only in the Working-Storage Section).
- OCCURS - Indicates that the item is repeated several times.
- INDEXED phrase - Indicates that the subject of this entry, or an entry subordinate to this entry, is to be referred to by indexing.

- COPY - Indicates that the entire predicate of the description is actually to be found in a location other than in this data description entry.
- REDEFINES - Indicates that the item occupies the same memory area as a prior item.
- USAGE - Specifies which of several possible machine formats applies to this item.
- RENAMES - Permits alternative, possibly overlapping, groupings of elementary items.
- SYNCHRONIZED - Indicates that the item is to have a special orientation to computer word boundaries.
- JUSTIFIED - Overrides the normal item alignment rules when other items are moved to an item described with this clause.
- KEY phrase - Specifies the location of a record, or a set of data items, that serve to identify the ordering of data.
- BLANK WHEN ZERO - Results in the blanking of an item when the value of that item is zero.

For an elementary item, the PICTURE clause can specify all format details; consequently, other clauses are usually unnecessary.

Every elementary item except an index data item may be said to have a picture; that is, it has a set of properties which are expressed in a PICTURE clause. Accordingly, this manual often refers to an item's PICTURE, without regard to the actual clauses used in its record description entry. Similarly, every item has a definite size, class, and usage, regardless of how it has been defined.

For a group item, the record description entry may omit all descriptive clauses, or it may include any of the following descriptive clauses:

VALUE OCCURS COPY REDEFINES	}	If any of these clauses are used in a group entry, the entire group is affected.
USAGE	}	If this clause is specified in a group entry, each subordinate elementary entry inherits the specified property.

It is essential that an item's data description be consistent with the values the item may actually assume. For example, an item whose values may actually be negative may be handled improperly in the object program if its description omits an operational sign. Similarly, an item described as numeric may be handled improperly if its values actually contain space characters. (In practical data processing applications, some situations are commonly encountered which are improper according to COBOL rules; one example is a numeric item with leading spaces instead of zeros.)

DISPLAY Item Formats

Any DISPLAY-n item occupies an integral number of adjacent internal character positions, and has no particular relationship to machine words except when the SYNCHRONIZED clause is specified in its record description entry. The significance of the respective DISPLAY-n usage is as follows:

- DISPLAY signifies that the item's actual machine format corresponds to the standard data format.
- DISPLAY-1 signifies that the item has an edited floating-point format (described below). The item's class is implicitly alphanumeric.
- DISPLAY-2 signifies that the commercial collating sequence is to be applied to the item.

For an item whose usage is DISPLAY or DISPLAY-1, the standard binary code is used within memory to represent each data character, except in the case of signed numeric items. If the value of a signed numeric item is nonnegative, the data characters employ the standard binary codes. If the value is negative, the sign is expressed by a variation in the binary code of the least significant digit; specifically, all bits except the 2^5 bit have the usual values, but the 2^5 bit is set to 1. (This convention corresponds to the punched card convention of no overpunch on nonnegative values, with an 'eleven' overpunch over the low-order digit of a negative value.)

The only respect in which the internal representation of DISPLAY-2 items differs from that of DISPLAY items is that a special six-bit binary code is used for each DISPLAY-2 character instead of the standard code. The special codes are chosen so that the result of comparing two DISPLAY-2 items is consistent with the commercial collating sequence rather than the machine's standard collating sequence.

The following special format is provided for DISPLAY-1 items. (This option is not a standard COBOL feature.)

$$\left\{ \begin{array}{c} + \\ - \end{array} \right\} 9.9(n)E \left\{ \begin{array}{c} + \\ - \end{array} \right\} 99$$

The first character is the report sign for the mantissa. The next characters represent the actual value of the mantissa; n may be a one- or two-digit integer from one to 17. The E represents an E insertion character, which is counted in the item's size. The remaining characters are the report sign for the exponent and the two 9s represent the exponent itself.

The value represented by a DISPLAY-1 item is equal to the mantissa value multiplied by the power of ten indicated by the exponent. This format is useful only for very large or very small values which, in a normal DISPLAY format, would begin or end with a long string of zeros. In particular, computations involving COMPUTATIONAL-2 items (see below) may sometimes produce results for which the DISPLAY-1 format is needed.

The format of a DISPLAY-1 item must be specified via the PICTURE clause. The item's size equals 7+n. For example, a DISPLAY-1 item whose PICTURE is +9.9(3)E+99 has size 10. If the value 0.000001724 were moved to this item, the result would be +1.724E-06.

COMPUTATIONAL Item Formats

The internal data formats for COMPUTATIONAL items are described below. COMPUTATIONAL item formats are used to obtain both internal and external space-saving and performance advantages.

The functions of the various COMPUTATIONAL options are:

- COMPUTATIONAL - Results in the decimal-precision format. This is the preferred usage for items involved in calculations within a processor that does not contain the Extended Instruction Set (EIS)¹.
- COMPUTATIONAL-1 - Results in the fixed-point binary integer format. This usage is applicable for items having only integral values, and is the preferred usage for items used as subscripts or referred to in DEPENDING options.
- COMPUTATIONAL-2 - Results in the floating-point binary format. This usage is appropriate for items whose absolute values may potentially exceed 10^{18} or be less than 10^{-18} ; it is also useful for data communication with non-COBOL programs.
- COMPUTATIONAL-3 - Results in the single-precision fixed-point binary integer format. This usage should be employed only for data communication with non-COBOL programs; even then, COMPUTATIONAL-1 or COMPUTATIONAL-2 should be used instead if the application permits.
- COMPUTATIONAL-4 - Results in the packed decimal format. This usage, available only with EIS processors, provides performance advantages but may require additional data space.

An implicit sign is assumed in the formats for all COMPUTATIONAL options.

When data items are described as COMPUTATIONAL or COMPUTATIONAL-2, ultimate accuracy for maximum length composite of operands (18 digits) in arithmetic statements may not be attainable due to floating-point hardware limitations.

COMPUTATIONAL-n items are not stored in a manner related to the character position subdivisions of a computer word. Instead, such items are stored as follows:

- COMPUTATIONAL and COMPUTATIONAL-2 items utilize the binary floating-point format.
- COMPUTATIONAL-1 items are stored as one-word or two-word binary integers.

¹ Extended Instruction Set (EIS) refers to an extension to the original Series 6000 instruction set. EIS is the standard instruction repertoire for models 6025, 6040, 6060, and 6080 of the Series 6000 system and for all models of the Series 60 system.

Since a stored COMPUTATIONAL-n item has no character orientation, an attempt to manipulate it as if it were made up of characters is meaningless. Thus, the storage area may be redefined for some distinct purpose but not, for example, to give separate access to the integral and fractional parts of the COMPUTATIONAL item. For similar reasons, a group MOVE statement involving COMPUTATIONAL-n items should normally entail only sending and receiving groups with similar descriptions; a MOVE CORRESPONDING statement should be used otherwise. COBOL rules do not require adherence to the suggestions given in this paragraph, but the user must assure that the application of a group MOVE statement or of a redefinition is legal.

BINARY REPRESENTATION OF FRACTIONAL VALUES

An important feature of the COMPUTATIONAL usage is a provision for the fractional part of an item. For a COMPUTATIONAL item, the fractional part and integral part are jointly represented as a binary integer, which corresponds exactly to the conceptual decimal value of the item. For a COMPUTATIONAL-2 item, however, the fractional part of the value is represented as a pure binary fraction of a limited number of bits. (A COMPUTATIONAL-1 item is an integer and therefore has no fractional part.)

A decimal fraction of a given number of digits cannot be represented exactly by a binary fraction of any finite number of bits. Consider, for example, the value 1/5, which is represented in decimal notation as 0.2. Trying to represent it by a four-bit binary fraction, one obtains $(.0011)_2$ or 3/16; with eight bits, one obtains $(.00110011)_2$ or 51/256. In fact, the exact value must be written as

$$(0.2)_{10} = (0.\overline{0011})_2$$

which means that the bit pattern 0011 in the binary expansion keeps repeating indefinitely. If the decimal value 0.2 is converted to a binary expansion of 71 bits and then converted back, the one-digit result would be 0.1, quite different from 0.2. The four-digit result would be 0.1999, which is almost (but not quite) equal to 0.2. If computations were involved instead of only conversions, the imprecision in the decimal result could be much greater.

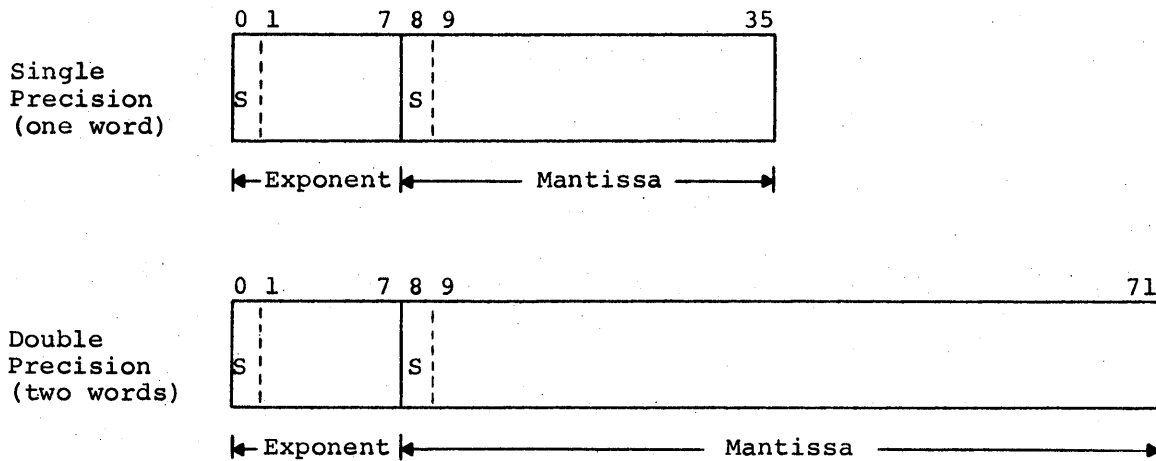
Various adjustments can be made to binary fractional values to make exact decimal results highly probable. The sure way, however, is to use binary integer notation to represent all values, whether integral or fractional. A consequence of doing so is that multiplication or division of an operand by a power of ten is sometimes necessary in the course of a computation. COMPUTATIONAL items use the equivalent of binary integer notation, and the compiler automatically supplies any required multiplications or divisions by powers of ten. (The formats and conventions governing COMPUTATIONAL items are described below.)

In most commercial data processing applications, particularly where dollars and cents are involved, a high degree of decimal precision is expected. For this reason, the COMPUTATIONAL and COMPUTATIONAL-1 usages are recommended over COMPUTATIONAL-2.

COMPUTATIONAL FORMATS

COMPUTATIONAL Data Items

The machine format for computational data items is the standard floating-point binary format (single or double precision):



COMPUTATIONAL items utilize the above format in a special way. The exponent indicates how many bits of significant information are present in the mantissa. Bits to the right of the point indicated by the exponent are not significant; these bits are normally all zero for COMPUTATIONAL items.

The sign of the exponent is normally nonnegative. The sign of the mantissa is the algebraic sign of the COMPUTATIONAL item.

The value stored in the mantissa is a binary integer, obtained as follows: if the item's data description specifies fractional places, the mantissa is stored as if the value had been multiplied by a sufficiently high power of 10 to make the value an integer. (The power of ten is called the 'span multiplier'.) Thus, 3.142 would be stored as 3142, as if it had been multiplied by 10^3 .

In general, if an item's picture is 9(p)V9(q), a suitable span multiplier is 10^q or any higher power of 10. If the span multiplier actually chosen is 10^s (with $s \leq q$), the s is called the 'span number'. The 'span' of a decimal-precision item is defined as the number of fractional digits permitted for an item in view of its span multiplier. Referring to the above example, 3.142 would be assigned 'span 3', allowing three fractional digits.

The significance of the conventions described is that a binary fraction or mixed number 'equivalent' to the decimal value could in general only be approximate, not exact, but the span multiplier permits the value stored to be exact in the decimal-precision format.

The compiler selects the span multiplier for each COMPUTATIONAL item and supplies appropriate coding to align the operands and the results in all computations. A 'span conversion' is sometimes required for this purpose; this means multiplication or division by a suitable power of 10 (always with a positive exponent). The general rules for COMPUTATIONAL item alignment are:

1. If a MOVE statement, or an addition or subtraction function, involves operands with the same span number, they are properly aligned without span conversion. Otherwise, one or more span conversions are necessary.
2. The span number of a product equals the sum of the span numbers of the operands, so span conversion via division is usually necessary to obtain proper alignment of the result.
3. The span number of a quotient equals the difference of the span number of the dividend and that of the divisor, so span conversion via multiplication is usually necessary to obtain proper alignment of the result.

Because of rule 1 above, it is desirable to have summands in the same span, to avoid span conversions. Here, another decimal-precision format convention becomes important; since a given item's span number can in principle be any number equal to or greater than the number of fractional places in the item's description, items with widely different PICTUREs can often be assigned the same span.

For example, span 3 in single-word precision can permit one to three fractional places and zero to five integral places. In double-word precision, span 3 can permit one to three fractional places and six to 15 integral places. Since items with no more than three fractional places are very common in commercial data processing applications, it is desirable to assign span 3 to items whenever possible, even if they have only one or two fractional places, to optimize their formats for addition and subtraction. This reasoning leads naturally to the concept of preferred spans.

To reduce span conversions, certain spans, each of which is applicable to a wide range of PICTUREs, are assigned preferentially. Thus, an item which could be assigned either to the span which is 'preferred 1' or to that which is 'preferred 2' would be assigned to the former. When the span has been assigned, it can be determined whether the item's PICTURE requires single or double precision, with single precision chosen whenever possible. The following rules apply for span and precision number assignments:

Single-Precision Items:

<u>Integral Places</u>	<u>Fractional Places</u>	<u>Span Number</u>	<u>Comment</u>
1-8	0	0	Preferred 1
0-5	1-3	3	Preferred 2
0-3	4-5	5	Preferred 3
0	8	8	Preferred 4

Double-Precision Items:

<u>Integral Places</u>	<u>Fractional Places</u>	<u>Span Number</u>	<u>Comment</u>
9-18	0	0	Preferred 1
16-17	1	1	
16	2	2	
6-15	1-3	3	Preferred 2
14	4	4	
4-13	4-5	5	Preferred 3
11-12	6	6	
11	7	7	
1-10	8	8	Preferred 4
0-10	6-7	9	
7-9	9	9	
7-8	10	10	
7	11	11	
0-6	9-12	12	Preferred 5
4-5	13	13	
4	14	14	
0-3	13-15	15	Preferred 6
1-2	16	16	
1	17	17	
0	16-18	18	Preferred 7

For the best results, it is recommended that the items in a program occur primarily in span 0 and span 3, and that the single-precision format be selected whenever possible. As shown in the preceding rules, most practical PICTURES that specify eight or less digits will result in single-precision formats. Utilizing these rules, the assignment of a given span and precision value can be forced by supplying an appropriate PICTURE.

The following examples illustrate the results of various span combinations:

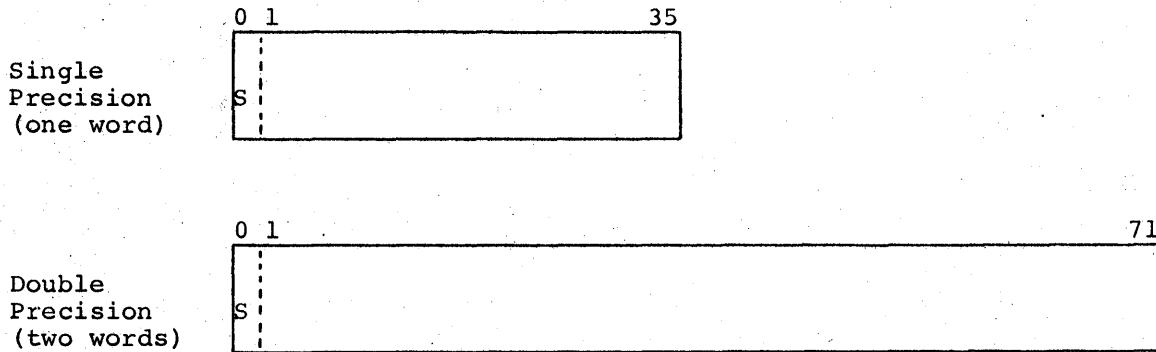
1. Given: A with PICTURE 9V99 and B with PICTURE 9(5)V9(3); the sum of A and B must be computed. The rules indicate that both are single-precision span 3 items; therefore, no span conversion is necessary.
2. Given: C with PICTURE 9(6) and D with PICTURE 9(6)V99. C is then a single-precision span 0 item and D is a double-precision span 3 item. If the sum of C and D is desired, a span conversion will be necessary. If their product, however, is to be a double-precision span 3 number, no span conversion is required for the multiplication.
3. Given: E with PICTURE 9(3)V9(3) and F with PICTURE 9(10)V9(3). Both are assigned span 3, but E is a single-precision item and F is a double-precision item. E and F may be added without conversion to produce a span 3 sum. Suppose, however, the product is to be stored in a span 3 item. The product of two span 3 numbers is a span 6 number. Therefore, a span conversion from span 6 to span 3 must follow the multiplication. Specifically, the span 6 product must be divided by 10^3 . If a span 3 quotient is desired, division of two span 3 numbers results in a span 0 quotient that must be converted to a span 3 number via multiplication by 10^3 .

If machine floating-point format is used for COMPUTATIONAL items, many operand alignment procedures occur automatically via floating-point hardware. Another convenience is that single- and double-precision operands can be mixed arbitrarily in a floating-point computation without the requirement for programmed conversions. A computation proceeds in double precision only when at least one of the operands is a double-precision item.

Another important advantage gained by using the floating-point format is that in a computation involving several arithmetic operations (resulting from a complex COBOL formula, for example), the hardware retains extra significant data in each intermediate step, so that the conceptual 18-digit limit on operands may occasionally be meaningfully exceeded on intermediate results. The overall significance, however, never exceeds 21 digits (so that the result of multiplying two 18-digit numbers, for example, cannot be 36 digits, even in an intermediate result). The 18-digit limit always applies to stored values.

COMPUTATIONAL-1 Data Items

The machine format for COMPUTATIONAL-1 data items is single- or double-precision fixed-point binary integer format:



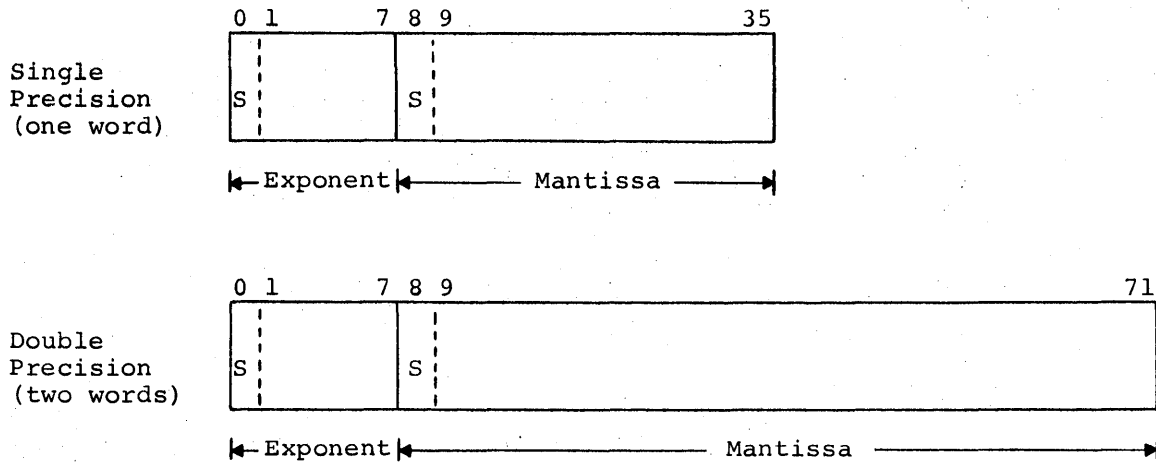
Although it is stored as a binary number, a COMPUTATIONAL-1 item's value is equal to the decimal value of the item because COMPUTATIONAL-1 items are restricted to integral values.

The precision assignment rules for COMPUTATIONAL-1 items are:

<u>Number of Digits in PICTURE</u>	<u>Precision</u>
1-8	Single
9-18	Double

COMPUTATIONAL-2 Data Items

The machine format for COMPUTATIONAL-2 data items is the single- or double-precision floating-point binary format:



The mantissa of a COMPUTATIONAL-2 item is a pure binary fraction and consequently is not necessarily exactly equivalent to the item's decimal value. The equivalence may be sufficiently close, however, for practical purposes.

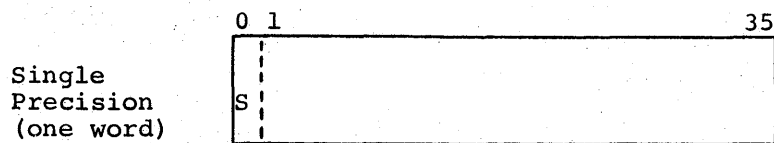
The COMPUTATIONAL-2 usage is especially effective for the operands in an elaborate formula. Should an operand value or an intermediate or final result exceed 10^{18} , or be less than 10^{-18} , only the floating-point binary format provides enough significance to yield meaningful results.

The precision assignment rules for COMPUTATIONAL-2 items are:

<u>Number of Digits in PICTURE</u>	<u>Precision</u>
1-8	Single
9-18	Double

COMPUTATIONAL-3 Data Items

The machine format for COMPUTATIONAL-3 data items is single-precision fixed-point binary integer format:



A COMPUTATIONAL-3 item is stored in the same format as a COMPUTATIONAL-1 item since both usages are restricted to integral values only. A COMPUTATIONAL-3 item, however, may contain a ten-digit integral value. This usage is intended to permit data communications with programs creating binary integer values which are single-precision numbers but are larger than eight integral digits.

COMPUTATIONAL-4 Data Items

The machine format for COMPUTATIONAL-4 data items is packed decimal format:

0	1	4	5	8	9	10	13	14	17	18	19	22	23	26	27	28	31	32	35
Z	0		1	Z	2		3		Z	4		5		Z	6		7		

Packed Decimal (4-bit)

The 'Z' in the bit positions represents the bit value 0 while other numbers in the fields represent the character positions.

A COMPUTATIONAL-4 item signifies that two four-bit digits are to occupy one nine-bit byte (packed decimal). Data using this format can be processed only on a computer that has the Extended Instruction Set (EIS) capability. If the PICTURE character-string specifies an operational sign, the item will be one four-bit digit larger than the number of '9's in the PICTURE character-string would imply.

The use of this data type may result in implicit character positions being allocated by the compiler since a COMPUTATIONAL-4 item may only start and end on a word or half-word boundary.

The packed decimal format can be defined independently or within a record that also contains six-bit Hollerith characters, binary integer fields, or floating-point fields. When files containing such records are written to tape handlers, extra care should be taken. The RECORDING MODE IS BINARY clause can be specified safely. If a mixed Hollerith/packed decimal file is written in the BCD or in the nine-track mode, bits will be lost from the data.

SECTION III

FILE DESCRIPTIONS

This section presents the language elements in the Environment Division and the Data Division that are used to describe files. Refer to the COBOL Reference Manual for the specific formats, rules, and special considerations applicable to these language elements.

SELECT SENTENCE

The SELECT sentence in the FILE-CONTROL paragraph of the Environment Division is used to name a file, identify the file medium, or describe some of the file properties. Each file named in each SELECT sentence must have a unique name.

OPTIONAL Phrase

The OPTIONAL phrase is used to indicate that the input file described will not necessarily be present each time the object program is executed. This feature can be utilized for describing files which may be present only during special periods such as the end of month or the end of year.

The OPTIONAL phrase may be specified only for input files which are to be accessed in a sequential manner. (Refer to the discussion of Sequential-Access Processing in Section V for additional information.)

OVERLAY Phrase

The OVERLAY phrase is used in run units that contain more than one COBOL object program. The OVERLAY phrase must be used whenever the same file-code is specified in more than one of the programs.

If more than one COBOL program in a run unit defines a file using the same file-code, the following restrictions apply:

1. The file definitions must be identical.
2. All but one of the file definitions must specify the OVERLAY phrase in the SELECT sentence of the FILE-CONTROL paragraph. This restriction applies even if no overlays are contained in the run unit.

If, during the execution of an object program, an overlay module is loaded into a memory location that is already occupied by a currently active input-output subroutine, the program may abort in an undefined manner. In this context, 'currently active' indicates that the subroutine is one of the routines being used to service the file. Therefore, the following two general prohibitions must be observed:

1. When a file is in the open state, no overlay module may be loaded into a memory location occupied by that file's file control information.
2. When a file is in the open state, no overlay module may be loaded into a memory location occupied by any of the subroutines that are servicing that file.

The file properties (including any RERUN phrases) must be identical in each program which is to reference the file.

Example:

Program A

```
SELECT OVERLAY FILE-A ASSIGN TO A0 FOR CARDS
RESERVE 1 ALTERNATE AREA.
```

Program B

```
SELECT FILE-A ASSIGN TO A0 FOR CARDS RESERVE 1 ALTERNATE AREA.
```

Although the two programs are described differently, the file properties of FILE-A are identical. Program B must have been loaded into memory to establish FILE-A's file properties before program A references FILE-A.

If one program is recompiled, all programs in the module overlay environment should be recompiled using the same Software Release version of the COBOL compiler to ensure that the file properties remain the same.

File-Name Phrase

Each file to be processed by the COBOL program must be named only once as a file-name following the keyword SELECT. Each selected file must have a file description (FD) entry or sort-merge file description (SD) entry in the Data Division, except when the RENAMING phrase is used.

Example:

```
.  
.  
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
    SELECT FILE-1 ASSIGN TO B1.  
    SELECT FILE-2 ASSIGN TO B2.  
    SELECT FILE-3 ASSIGN TO B3.  
DATA DIVISION.  
FILE SECTION.  
FD FILE-3 LABEL RECORD IS STANDARD.  
.  
.  
SD FILE-2.  
.  
.  
FD FILE-1 LABEL RECORD IS OMITTED.  
.  
.
```

Although each selected file must have an FD or SD described in the Data Division, the order of description need not be the same for each file.

RENAMING Phrase

The RENAMING phrase provides a shorthand method of describing the same file twice. This feature can be useful when a COBOL program requires two identical descriptions of a file for purposes such as updating a master file.

When the RENAMING phrase is used, the COPY option must be included on the \$ COBOL card and the LIBCOPY option must not be included on the \$ COBOL card.

When the RENAMING phrase is used, the file description (FD) entry and related record description entries associated with the file-name being renamed are applied to the renaming file; therefore, the latter must not be described in the File Section of the Data Division.

Example:

```
.  
.  
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
    SELECT ABC ASSIGN TO C1.  
    SELECT DEF RENAMING ABC ASSIGN TO C2.  
    SELECT GHI ASSIGN TO XY.  
DATA DIVISION.  
FILE SECTION.  
FD ABC LABEL RECORD IS STANDARD.  
01 REC-A PIC X(80).  
FD GHI LABEL RECORD IS OMITTED.  
01 REC-B PIC X(100).  
WORKING-STORAGE SECTION.  
.  
.
```

The SELECT sentence for the renamed file must not contain a RENAMING phrase. The renamed file must not have a sort-merge file description. The file description for the renamed file must not be the last file description entry in the File Section of the Data Division.

ASSIGN Phrase

When the object program is submitted for execution, it is accompanied by peripheral assignment cards which are used to specify the peripheral devices for each file. The file-code in the peripheral assignment card must be the same as that assigned by the COBOL object program.

Each file named in a SELECT sentence must be assigned to a peripheral device by specifying the file-code option in the ASSIGN phrase.

The file-code option must be a two-character word consisting either of two letters or one letter and one digit.

Example:

```
SELECT FILE-T ASSIGN TO I4.
```

The integer-1 option, which is intended to indicate the number of input-output units assigned to a given file-name, is treated as documentation since the assignment of multiple devices is handled by the operating system using control cards.

Integer-1 must not be specified when file-code-2, file-code-3, ..., is also specified in the ASSIGN phrase.

Multiple file-codes in the ASSIGN phrase are treated as documentation only.

File-codes beginning with the letter S, such as S1, S2, ..., SA, SB, ..., should not be used in programs utilizing either the sort or the merge feature, since these file-codes have a special meaning in the sort or the merge operation.

A COBOL reserved word, such as NO or OR, must not be specified as a file-code.

If the ACCESS MODE phrase is not specified in conjunction with the file-code option, the following statements are applicable:

1. If the physical record size is not specified by a BLOCK CONTAINS clause in the Data Division file description entry associated with this file, it will be presumed to be 320 words.
2. An implicit process area, block serial number, or variable-length record may not be presumed for the file; therefore, the appropriate APPLY phrase must be specified in the I-O-CONTROL paragraph for the option desired.

If the ACCESS MODE IS RANDOM phrase is specified in conjunction with the file-code option, the following statements are applicable:

1. The file must be assigned to a randomly addressed mass storage file space at object program execution. Refer to the \$ FILE card in the Control Cards reference manual.
2. If the physical record size specified in a Data Division file description entry BLOCK CONTAINS integer CHARACTERS clause exceeds the size of the logical record, the space between the end of the logical record and the end of the physical record will not be utilized. This area is referred to as padding.
3. A process area will be implicitly reserved for the file. Thus, the APPLY PROCESS AREA phrase need not be specified in the I-O-CONTROL paragraph.
4. The ACTUAL KEY IS phrase must be specified. The value of the data item referenced as the actual key must indicate the relative position of the logical record within the file, starting with the value zero for the first record.

If the ACCESS MODE IS SEQUENTIAL phrase is specified in conjunction with the file-code option, the following statements are applicable:

1. The file must be assigned to a linked mass storage file space at object program execution.
2. SYSTEM STANDARD FORMAT (I-O-CONTROL paragraph) must be specified, explicitly or implicitly.
3. A process area will be implicitly reserved for the file. Thus, the APPLY PROCESS AREA phrase need not be specified in the I-O-CONTROL paragraph.
4. The ACTUAL KEY IS phrase need not be specified. However, the contents of the data item referenced as the actual key will be updated when the ACTUAL KEY phrase is specified.

FOR CARDS Phrase

To avoid format errors, output files intended for eventual punching in card form by either system output (SYSOUT) or by Bulk Media Conversion must be identified as such by specifying the FOR CARDS phrase. The file format will be presumed to be the Series 60/6000 system standard format and each logical record will be assigned the Hollerith card image media code. The direct allocation of a card reader or card punch to a COBOL program will have an adverse affect on system performance and is not recommended.

When the FOR CARDS phrase is specified, the compiler will automatically apply a process area to the file.

FOR LISTING Phrase

To avoid format errors, output files intended for eventual printing by either system output (SYSOUT) or by Bulk Media Conversion must be identified as such by specifying the FOR LISTING phrase. The file format will be presumed to be the Series 60/6000 system standard format and each logical record will be assigned the Hollerith print line media code. The direct allocation of a printer to a COBOL program will have an adverse affect on system performance and is not recommended.

When the FOR LISTING phrase is specified, the compiler will automatically apply a process area to the file.

Printer advancement control characters and a report code are automatically provided for each print line.

Since current printer hardware will process print lines containing as many as 160 columns, it is no longer practical for the compiler to perform compile time checks on maximum print line sizes for the various printers that may be configured on the system. Therefore, the user is responsible for allocating printers that are compatible with the files assigned to them. For example, if a production COBOL program that requires a 160-column printer is assigned a 132-column or 136-column printer, data alerts should be expected.

FOR MULTIPLE REEL/UNIT Phrase

In Series 60/6000 COBOL, the MULTIPLE REEL/UNIT options are treated as documentation only.

RESERVE Phrase

The RESERVE phrase allows the user to modify the number of input-output memory areas allocated by the compiler.

If the RESERVE phrase is omitted, the compiler automatically allocates two buffer areas for file processing.

INTEGER OPTION

If integer is specified, the compiler will assign a maximum of two buffer areas.

Example:

```
SELECT FILE-G ASSIGN MN RESERVE 3 ALTERNATE AREAS.
```

The compiler will reserve two input-output buffer areas for FILE-G.

NO OPTION

If the RESERVE phrase is used and NO is specified, one input-output buffer area will be reserved by the compiler.

FOR BLANK COMMON Phrase

Although buffer space is normally allocated to the Labeled Common storage area, it is possible, by specifying the FOR BLANK COMMON phrase, to force the allocation of buffer space to the Blank Common storage area. A run unit containing more than one object program that utilizes the Blank Common feature may require explicit control card directives to ensure the correct positioning and extent of the Blank Common storage area. Refer to the description of the \$ LOWLOAD card in the General Loader reference manual for the operational characteristics of the Labeled Common and Blank Common storage areas.

The FOR BLANK COMMON phrase must not be used in any program that requires segmentation. It can, however, be used with caution in a module overlay environment program in which files using this feature may be common to another program. If a file common to two or more programs is to be assigned to Blank Common storage, the file must be so assigned in each program in which it is common. An identical ordering of files by SELECT sentences is required in each program when more than one such file is involved.

FILE-LIMIT(S) Phrase

The FILE-LIMIT(S) phrase in the FILE-CONTROL paragraph is included for program documentation only. The actual limits are established by the file space allocated by the file control cards for the run unit.

Attempts to access a logical record outside the logical segments of the file will result in the execution of either the AT END or INVALID KEY phrase.

ACCESS MODE Phrase

The ACCESS MODE phrase must be specified for files that will be assigned to a mass storage device at object program execution.

When ACCESS MODE IS SEQUENTIAL is specified, the mass storage logical records are read or written starting with the first record in the file and proceeding to the last record in the file. Refer to the discussion of Sequential-Access Processing in Section V for additional information.

When ACCESS MODE IS RANDOM is specified, the mass storage logical records are read or written in the order indicated by the contents of the data-name designated in the ACTUAL KEY phrase. Refer to the discussion of Random-Access Processing in Section V for additional information.

PROCESSING MODE Phrase

The PROCESSING MODE phrase must be specified for mass storage files to indicate that the logical records are to be processed in the order in which they are accessed.

ACTUAL KEY Phrase

The ACTUAL KEY data item must be a single-precision binary integer described with USAGE COMP-1. In addition, the ACTUAL KEY data item must be described as either a level 01 or level 77 data description entry in the Working-Storage Section of the Data Division.

The ACTUAL KEY phrase is required for mass storage files for which the ACCESS MODE IS RANDOM phrase is specified. The ACTUAL KEY phrase identifies that data item whose value controls the access to the logical records on mass storage file space. When a READ, WRITE, or SEEK statement is executed, the ACTUAL KEY data item contains the integer value of the ordinal record position that is to be accessed.

The ACTUAL KEY phrase may be optionally specified for a file for which the ACCESS MODE IS SEQUENTIAL phrase is specified. In this case, the data item identified in the ACTUAL KEY phrase does not control the access to the mass storage file space. However, the ACTUAL KEY data item will be updated during the execution of a READ or WRITE statement with the integer value that represents the ordinal record position of the logical record accessed by the program. In particular:

1. If the last input-output action was an OPEN statement, the actual key value represents the lowest accessible ordinal record position on the file.
2. If the last input-output action was a successful READ statement, the actual key value represents the ordinal record position of the record obtained.
3. If the next input-output action may legitimately be a WRITE statement, the actual key value represents that ordinal record position into which the WRITE statement will attempt to place a record.
4. The actual key value associated with the execution of either the AT END phrase of a READ statement or the INVALID KEY phrase of a WRITE statement represents the ordinal record position at which the exception condition was detected.
5. The CLOSE statement has no effect upon the actual key value.

The first ordinal record position on a sequential mass storage file is associated with an actual key value of one (1). The first ordinal record position on a random mass storage file is associated with an actual key value of zero (0). However, even when an actual key value of zero is available, programmed use of the value one (1) as the lowest actual key is preferred, since it allows a simple correspondence between subscript (and index) values and the ordinal record positions on a file.

APPLY PHRASE

The APPLY phrase in the I-O-CONTROL paragraph of the Environment Division is used to specify special input-output techniques which are to be applied to files defined in the FILE-CONTROL paragraph.

Contradictory input-output techniques must not be specified for a file-name. Refer to the Summary of File Property Relationships in Section V.

PROCESS AREA Phrase

The APPLY PROCESS AREA phrase can be used to increase object program efficiency for files which have heavy processing activity and are described either as having more than one logical record per physical record with the BLOCK CONTAINS clause in the file description entry or at least one alternate input-output area with the RESERVE phrase in the SELECT sentence of the FILE-CONTROL paragraph.

When the APPLY PROCESS AREA phrase is included, each logical record of an input file is moved to the 'process area' for processing when it is read and each logical record of an output file is developed in the 'process area' and moved to the buffer when it is written. Otherwise, the standard method is to process both input and output files in the buffer area.

The APPLY PROCESS AREA phrase may be redundantly specified even though an implicit process area has been reserved, by using another option such as the FOR CARDS or FOR LISTING option.

BLOCK SERIAL NUMBER Phrase

The APPLY BLOCK SERIAL NUMBER phrase indicates that each physical record is to be prefixed with a word containing the relative position of the physical record within the file.

The SYSTEM STANDARD FORMAT option imposes block serial numbers. In addition, block serial numbers may optionally be used on files which do not have the system standard format, provided that the recording mode is binary.

Sort files and merge files are implicitly given block serial numbers; therefore, the APPLY BLOCK SERIAL NUMBER phrase would be redundant for files defined with a sort-merge file description entry in the Data Division.

The explicit or implicit specification of the APPLY BLOCK SERIAL NUMBER phrase in the source program must match the physical presence or absence of block serial numbers on the file. If there is no match between the file description and its physical format, an error condition will result when that file is read.

When applied to a file, block serial numbers use the first computer word of each physical record. This word has the following format:

0	17 18	35
block serial number (binary integer)	block size in words (binary integer)	

The block serial number is the sequential number of this physical record within the current reel of this file (the current reel pertains only to magnetic tape files).

The block size is the actual size of this physical record, excluding the block serial number control word itself.

COBOL procedural statements cannot access the block serial number control word.

SYSTEM STANDARD FORMAT Phrase

The APPLY SYSTEM STANDARD FORMAT phrase provides a shorthand method for describing the file properties of a file which may be processed on different hardware devices each time the object program is to be executed. Refer to the description of system standard format in Section V for further details.

The Data Division file description entry clauses must not contradict the file properties having the system standard format.

VLR FORMAT Phrase

The APPLY VLR FORMAT phrase causes the logical records of the file to be preceded by a record control word which contains the record size in words and other control information. The recording mode will be presumed to be binary.

Depending on the BLOCK CONTAINS, RECORD CONTAINS, and LABEL RECORD clauses of the file description entry, the VLR format may or may not apply to a file that conforms to the system standard format.

RERUN Phrase

The RERUN phrase causes checkpoint memory dumps to be taken. That portion of the phrase used is RERUN ON file-name-7 EVERY integer-1 RECORDS OF file-name-8.

If 'ON file-name-7' is specified, the output device allocated to file-name-7 receives the checkpoint dump; otherwise, the output device allocated to file-name-8 receives the checkpoint dump. If 'ON file-name-7' is specified, file-name-8 may be either an input or an output file.

The number of records specified by integer-1 may not exceed 250,000.

The output device must be opened as an output file at every point in the program where a READ or a WRITE statement references file-name-7, if specified, or file-name-8 so that the output device can receive the checkpoint dump.

SAME AREA Phrase

The SAME AREA phrase indicates that two or more files are to use the same memory area during object program execution. The memory area to be shared includes all storage areas and alternate input-output areas assigned to the referenced files.

Only one of the files may be opened at a time.

A file-name must not be used in more than one SAME AREA phrase.

SAME RECORD AREA Phrase

The SAME RECORD AREA phrase indicates that two or more files are to use the same memory area for processing the current logical record and implies a process area for the named files. All the files named in the SAME RECORD AREA phrase can be open at the same time. Each logical record processed in the record area is considered to be the current logical record of each file named. A file may be specified in only one SAME RECORD AREA phrase.

There is no requirement that the logical record descriptions be identical for each file that shares the same record area; however, undesirable results could occur if their record descriptions were different and this feature were not used with discretion.

SAME SORT or SORT-MERGE AREA Phrase

The SAME SORT-MERGE AREA phrase is equivalent to the SAME SORT AREA phrase and both are treated as documentation only. The space used by the sort or merge process is determined dynamically at execution time.

MULTIPLE FILE Phrase

The MULTIPLE FILE TAPE phrase is required when two or more files share the same reel of tape.

Only those files on a multiple file tape that are referenced in the source program need be named in a MULTIPLE FILE phrase.

If all files on the tape are referenced in the order in which they appear on the tape, the POSITION option may be omitted.

Example:

```
.  
INPUT-OUTPUT SECTION.  
FILE-CONTROL.  
  SELECT FILE-C ASSIGN TO T1.  
  SELECT FILE-A ASSIGN TO T2.  
  SELECT FILE-B ASSIGN TO T3.  
I-O-CONTROL.  
  SAME AREA FOR FILE-B FILE-C FILE-A.  
  MULTIPLE FILE TAPE CONTAINS FILE-A FILE-B FILE-C.
```

Although the files may be referenced in other phrases differently than they appear on the tape, they must be listed in their exact consecutive order in the MULTIPLE FILE phrase when the POSITION option is omitted.

If any file in the sequence of files on the tape is not included in the MULTIPLE FILE phrase, then the position relative to the beginning of the tape of each file named in the phrase must be given.

Example:

```
MULTIPLE FILE FILE-C POSITION 3 FILE-E POSITION 5  
  FILE-F POSITION 6 FILE-Z POSITION 26.
```

All of the files on a multiple file tape must have labels present or, conversely, all of the files must have labels omitted.

Example:

```
I-O-CONTROL.  
MULTIPLE FILE FILE-1 FILE-2.  
MULTIPLE FILE FILE-3 FILE-4.  
DATA DIVISION.  
FILE SECTION.  
FD FILE-1 LABEL RECORDS ARE STANDARD.  
01 REC-1 PIC X(320).  
FD FILE-2 LABEL RECORD IS STANDARD.  
01 REC-2 PIC X(320).  
FD FILE-3 LABEL RECORD IS OMITTED.  
01 REC-3 PICTURE X(132).  
FD FILE-4 LABEL RECORD IS OMITTED.  
01 REC-4 PICTURE X(80).
```

Each MULTIPLE FILE phrase describes one multiple file tape. In the preceding example, two multiple file tapes are described. There can be any number of multiple file input or output tapes (each having a corresponding MULTIPLE FILE phrase); however, all files listed for each tape must be contained on a single reel.

Only one file of a multiple file tape can be open at any given time.

Files referenced in a MULTIPLE FILE phrase cannot be described with the OPTIONAL phrase of the SELECT sentence in the FILE-CONTROL paragraph.

All files on a multiple file tape must have the same recording mode.

FILE DESCRIPTION ENTRIES

The File Section header is followed by a file description entry or a sort-merge file description entry.

A file description entry consists of a level indicator (FD), followed by a data-name (the name of the file) which corresponds to a data-name specified in a SELECT sentence in the FILE-CONTROL paragraph of the Environment Division, and a series of independent clauses. The clauses specify the manner in which the data is to be recorded on the file, the size of the logical and physical records, the names of the label records contained in the file, the values of specific label data items, and the names of the data records which compose the file.

If the RENAMING option has been specified in a file's SELECT sentence in the Environment Division, its description is implicitly provided, and must not be given explicitly in the File Section. When the RENAMING option is used, COPY must be included on the \$ COBOL card and the LIBCOPY option cannot be used. All other files require explicit descriptions in the File Section.

If a file is intended to have system standard format, the LABEL RECORD(S) IS/ARE STANDARD clause is recommended for its file description entry. The DATA RECORD(S), VALUE OF, and/or REPORT(S) clauses may be used optionally. When the BLOCK CONTAINS and RECORDING MODE clauses are omitted, the system standard format is assumed.

SORT-MERGE FILE DESCRIPTION ENTRIES

For sort or merge file description entries, the level indicator SD is followed by a data-name (the name of the file) which corresponds to a data-name specified in a SELECT sentence and a series of independent clauses. The clauses specify the name, size, and number of data records in the sort or merge file. A sort or merge file is a set of records to be ordered in ascending or descending sequence based on the specification of keys in a SORT or MERGE statement in the Procedure Division. No label procedures are under the control of the user and the rules for blocking and internal storage are peculiar to the SORT or MERGE statement.

Level Indicator and File-Name

The level indicator identifies the entry as a file description entry (FD), a sort-merge file description entry (SD), or a report description entry (RD).

The file-name identifies the file for subsequent references in the Environment Division and the Procedure Division. The file-name is the highest level qualifier available for data-names belonging to the file.

If the level indicator SD is used, the file-name must be the name associated with a sort-merge file. Except when used as a qualifier, the sort-merge file-name can appear in the Procedure Division only in SORT, MERGE, and RETURN statements.

Refer to Section VIII, Report Writer, for a detailed explanation of the report description entry (RD).

BLOCK CONTAINS Clause

The BLOCK CONTAINS clause is optional and may be omitted when the file has the standard physical record size of 320 computer words. When system standard format is intended, the clause should be omitted.

If integer-1 and integer-2 are both specified in the BLOCK CONTAINS clause, they refer to the minimum and maximum size of the physical record, respectively. In this case, integer-1 is understood to be for documentation only. If only integer-2 is specified, it represents the exact size of the physical record. Integer-1 (when used) and integer-2 must be unsigned nonzero integers.

Regardless of whether the physical record size is given in terms of characters or records, each logical record will begin in a new computer word.

Whenever the keywords RECORDS or CHARACTERS are not specifically written in the clause, the CHARACTERS option is presumed.

When the CHARACTERS option is used, the physical record size is specified in terms of the number of standard characters contained in the physical record, regardless of the types of characters used to represent the items within the physical record.

When the SYSTEM STANDARD FORMAT option is designated, explicitly or implicitly, the compiler allocates a physical record size of 320 computer words. If the CHARACTERS option is specified, integer-2 must not exceed a value of 1920.

Unless system standard format is indicated, explicitly or implicitly, a magnetic tape file may have any desired physical record size not exceeding 4095 words.

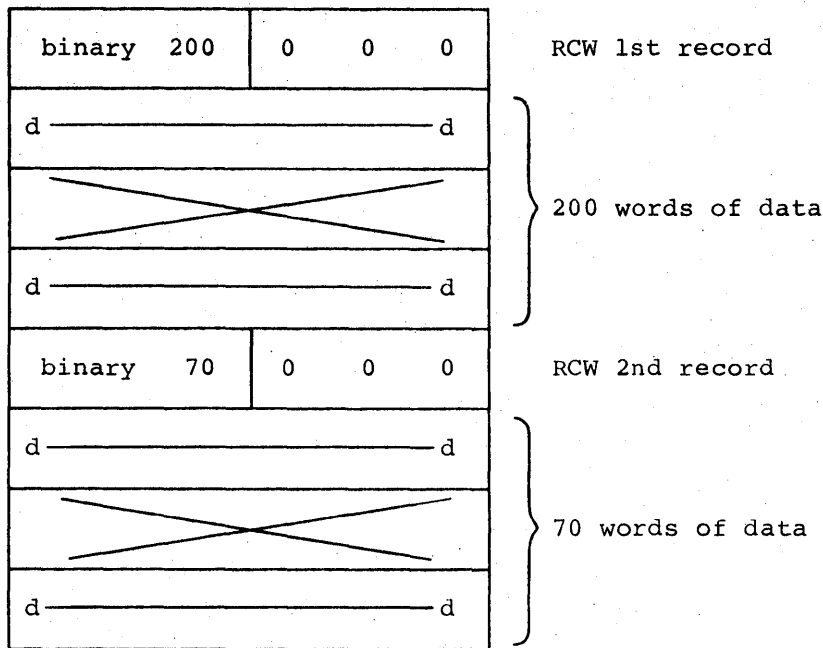
Calculation of the physical record size for sequential-access files must reflect the basic record format (fixed or variable) and the number of logical records in the physical record. The physical record size conventions, in terms of the number of computer words, are presented below. The formulas yield 'net' physical record sizes, and do not reflect block serial numbers. If block serial numbers are desired, the 'net' physical record size must be increased by one word.

There are three cases for fixed-length record (FLR) files:

1. BLOCK CONTAINS clause omitted. The overall block size will not exceed 320 words. Net block size is the largest multiple of the record size not exceeding 320 words (319 words if block serial numbers are applied). If an FLR file has 30-word records, net block size (with the BLOCK CONTAINS clause omitted) would be 300 words.
2. BLOCK CONTAINS integer-2 RECORDS. The net block size is integer-2 times the record size. In the example given for case 1 above, the same result would be obtained by specifying BLOCK CONTAINS 10 RECORDS.
3. BLOCK CONTAINS integer-2 CHARACTERS. The block size will be as close to integer-2 divided by six words as possible. The net block size is the largest multiple of the record size not exceeding (integer-2 divided by six). In the example given for case 1 above, the same result would be obtained by specifying BLOCK CONTAINS 1800 CHARACTERS. Integer-2 should be exactly six times the overall record size in words, multiplied by the desired blocking factor, plus six if block serial numbers are applied.

There are also three cases for variable-length record (VLR) files:

1. BLOCK CONTAINS clause omitted. The maximum block size is 320 words (including block serial number, if applied). The actual block size will vary from one block to the next. In each output block, successive records are added until the remaining space is insufficient to hold another record; the current block is then physically written out and a new block is begun. The physical block ends with the last word of the last record. If record sizes are 200 words and 70 words, respectively, a block might appear as shown in the following diagram.



Thus, the actual block size in this example is $(1 + 200) + (1 + 70) = 272$ words.

2. BLOCK CONTAINS integer-2 RECORDS. The maximum net block size is (maximum record size in words + 1) times integer-2. The increment of one (1) to maximum record size allows for record control words. This convention allows at least integer-2 records per block, but the blocks are still constructed as described for case 1 (under VLR files) above. If not all records in a block are of the maximum size, more than integer-2 records may be included.

If the file has two record types, with sizes (in words) of 5 and 75, respectively, and BLOCK CONTAINS 2 RECORDS, maximum block size is $2 \times (75 + 1) = 152$; the block might, however, contain as many as 25 of the small records.

3. BLOCK CONTAINS integer-2 CHARACTERS. The maximum block size is the largest integer not exceeding (integer-2 divided by six), including block serial number, if applied. Integer-2 should be a multiple of six. Blocking conventions proceed as described for case 1 (under VLR files) above.

The use of the word CHARACTERS in the clause is optional. Whenever the keyword RECORDS is not specifically written in the clause, the word CHARACTERS can be omitted with the understanding that integer-1 (if used) and integer-2 represent the number of characters in the block. The word RECORDS cannot be used for a file for which REPORT(S) is also specified.

When the word RECORDS is used with variable-length records, the block size is equal to the maximum record size (in computer words) multiplied by the number of records plus one.

For mass storage files assigned to a file-code and described with ACCESS MODE IS RANDOM, the physical record size associated with the CHARACTERS option is considered identical with the logical record size. Depending on the mass storage device normally intended for the file, the physical record size may be adjusted for efficiency by using the BLOCK CONTAINS clause with the CHARACTERS option. Integer-2 may range from 384 to 24,570 characters (which results in record sizes in the range 64 to 4095 words). A size that is not modulo 384 characters will result in wasted space.

Regardless of which BLOCK CONTAINS option is specified, the input-output system automatically adjusts to single record blocks at program execution if the file is actually assigned to a printer, remote terminal, card reader, or card punch.

DATA RECORD(S) Clause

The data-names specified in this optional clause must be level 01 items whose record description entries, together with their subordinate entries, follow the FD or SD entry. Standard label record-names must not be mentioned in this clause. If record descriptions for label records within an FD are included, they must appear between the FD entry and the data record descriptions.

The presence of more than one data-name indicates that the file contains more than one type of data record. If the record sizes in words are not equal, the file will be assigned the variable-length record format and its recording mode must be binary. The records of the file are not required to have the same description.

Example:

```
FD FILE-Y LABEL RECORDS ARE STANDARD DATA RECORDS ARE REC-Y
      REC-W REC-X.
01 REC-W PIC X(80).
01 REC-X.
   02 DN-1 PIC 9(6) COMP.
   02 DN-2 PIC 9(18).
   02 DN-3 PIC 9(18).
   02 DN-4 PIC 9(18).
   02 DN-5 PIC X(20).
01 REC-Y PIC X(100).
```

Conceptually, all data records in a file share the same area. This concept is not altered by the presence of more than one type of data record in a file. In the preceding example, REC-W, REC-X, and REC-Y would be implicit redefinitions of the same area whose size would be 100 characters.

The order in which the record description entries occur as 01 entries is not significant, with the exception of sort files. For a sort file with more than one size data record description, the first record description entry after the SD entry is assumed to be the dominant type. Its size is considered to be the most common in the sort file. Sort optimization is based on this assumption. Therefore, a careful choice in ordering record description entries for a sort file enhances object program efficiency.

LABEL RECORD(S) Clause

The LABEL RECORD(S) clause is the only required clause within the file description (FD) entry. This clause is used to designate whether or not labels are associated with the file and, therefore, whether or not the input-output system must be prepared to process labels for the file. A label record is a special type of logical record that contains information about the file or about the medium on which the file is recorded. Label records are not part of the data area of a file but appear as separate records at the logical beginning and ending of a file. On some media, such as magnetic tape, the label records are physically recorded preceding and following the physical beginning and ending of the file. On other media, such as mass storage, the label records might not be physically adjacent to the file.

On a magnetic tape, label records are separated from data records by a file mark and successive files are separated by a file mark. On a multiple file tape, file positioning is accomplished by counting the number of file marks passed. Therefore, the labels must be consistent for all of the files on such a tape. That is, either every file on the tape must be labeled or all of the files on the tape must be unlabeled. This is true whether or not all of the files are referenced in the COBOL program. Those files on a multiple file tape that are referenced in the program must have consistent descriptions. That is, either all of the referenced files must have the OMITTED option specified or each referenced file must specify either the STANDARD option or the label-name option.

OMITTED OPTION

The use of the OMITTED option signifies that no labels exist for the file. The input-output system will not expect to process labels for the file and, on input, will not check for the possible existence of a label.

The OMITTED option must not be specified if system standard format has been assigned for a file, explicitly or implicitly. System standard format invokes special processing conventions that presume labels, even though the physical device to which the file is assigned does not support the physical recording of labels.

If the OMITTED option is specified on an output file, each output tape is terminated with an end-of-file mark when the end-of-tape foil is detected and a tape swap occurs.

If the OMITTED option is specified on an input file, the standard means of recognizing the logical end of the file is not available. Unless the file is a single-reel file, the user must determine via explicit procedures which reel terminates the file. When an end-of-file mark is detected, the READ statement's AT END procedure is executed. If a subsequent READ statement is executed, a tape swap takes place and the first record of the next reel is obtained. If the reel just ended is the last reel on the file, the file should be closed.

STANDARD OPTION

The use of the STANDARD option signifies that logical labels that conform to the Series 60/6000 label format specifications are considered to exist for the file even though they may not be recorded on some of the physical devices to which the file may be assigned.

The STANDARD option must be specified if system standard format has been assigned for a file, explicitly or implicitly.

When the STANDARD option is specified, the Series 60/6000 standard beginning label format is inferred automatically by the compiler and consists of the following group item structure:

```
01 (fixed label-name)
02 LABEL-IDENTIFIER PICTURE X(12).
02 INSTALLATION PICTURE X(6).
02 REEL-SERIAL-NUMBER PICTURE BX(5).
02 FILE-SERIAL-NUMBER PICTURE BX(5).
02 REEL-NUMBER PICTURE BB9999.
02 DATE-WRITTEN.
03 LABEL-YEAR PICTURE B99.
03 LABEL-DAY PICTURE 999.
02 FILLER PICTURE BBB.
02 RETENTION-PERIOD PICTURE 999.
02 IDENTIFICATION PICTURE X(12).
02 FILLER PIC X(24).
66 ID RENAMES IDENTIFICATION.
```

The items within Series 60/6000 standard beginning labels have the following significance:

1. The implied value of LABEL-IDENTIFIER is ~~GE~~6000~~BT~~.
2. INSTALLATION contains constant information for each user installation. This item is supplied automatically in output labels, but is ignored by input label checking routines.
3. REEL-SERIAL-NUMBER contains the serial number of the physical tape reel. This number is also recorded externally on the reel itself. This item is supplied automatically in output labels.
4. FILE-SERIAL-NUMBER contains the serial number of the first reel of the file. On the first reel, therefore, the values of FILE-SERIAL-NUMBER and REEL-SERIAL-NUMBER are identical. This item is supplied automatically in output labels. On input, it may be checked against an expected value.
5. REEL-NUMBER contains the number of the reel within the file. The first reel is number 0001, the second is 0002, etc. This item is automatically supplied in output labels and checked on input.
6. DATE-WRITTEN contains the day of the year on which the object program has been executed to produce this file. This item is automatically supplied in output labels, but is ignored on input.
7. RETENTION-PERIOD contains the number of days the file is to be retained. This item is processed in two distinct phases on each output file:
 - (a) Every tape upon which an output file is to be written is expected to have a prior label, which may be either a blank reel label or a beginning label on which the RETENTION-PERIOD has expired (current date minus DATE-WRITTEN exceeds RETENTION-PERIOD). These conditions are checked and operator action is requested if they are not met.

- (b) If a value has been specified for the RETENTION-PERIOD via the VALUE OF clause in the FD entry, this value is automatically supplied in the new output label, which replaces the prior beginning label on the output tape. When RETENTION-PERIOD appears in the VALUE OF clause, a numeric literal not exceeding 999 must be specified. The value 999 signifies permanent retention.
8. IDENTIFICATION or ID contains a 12-character name assigned to the file for external identification. On input, this item is automatically checked against the value specified in the VALUE OF clause, if this clause is present in the FD entry. On output, the value specified in the VALUE OF clause, if present, is automatically supplied in the output label. If the VALUE OF IDENTIFICATION clause is not specified, this field will be checked against the value of the file-name field on the file assignment control card. If neither the VALUE OF clause nor the file-name field is specified, the identification field is ignored.

The beginning tape label and the beginning file label have identical formats. The fixed label-name for a beginning file label is BEGINNING-FILE-LABEL. The fixed label-name for a beginning reel label is BEGINNING-TAPE-LABEL.

When the STANDARD option is specified, the Series 60/6000 standard ending label format is inferred automatically by the compiler and consists of the following group item structure:

```
01 (fixed label-name)
02 SENTINEL PICTURE X(6).
   88 END-OF-TAPE VALUE IS "EOFF".
   88 END-OF-FILE VALUE IS "EOF".
02 BLOCK-COUNT PICTURE 9(6).
02 FILLER PICTURE X(72).
```

The items within Series 60/6000 standard ending labels have the following significance:

1. The END-OF-TAPE condition of SENTINEL causes an automatic tape swap on input.
2. The END-OF-FILE condition of SENTINEL signifies the end of the file.
3. BLOCK-COUNT is always described with USAGE COMPUTATIONAL-1, except when the recording mode is BCD; in this case, BLOCK-COUNT is implicitly USAGE DISPLAY, as shown above. BLOCK-COUNT is automatically supplied in output labels and is checked on input labels against the computed block count.
4. An ending tape label is automatically produced on output tapes when an end-of-tape foil is encountered. It may also be produced by a CLOSE REEL statement. The production of this label is automatically followed by a tape swap.
5. A CLOSE statement (without the REEL option) causes an ending file label to be produced, together with any other CLOSE statement options that may be specified.

The ending tape label and the ending file label have identical formats. The fixed label-name for an ending file label is ENDING-FILE-LABEL. The fixed label-name for an ending reel label is ENDING-TAPE-LABEL.

It is not necessary to describe the formats of the Series 60/6000 standard beginning and ending label records in the source program, since the compiler infers their descriptions when the STANDARD option is specified. However, when the STANDARD option is used, these descriptions may also be included as record descriptions in the source program by utilizing the four fixed label-names identified above. These record descriptions must not be named in the DATA RECORDS clause. When such explicit label record descriptions are specified, the standard label contents must be described exactly as shown in the above formats with respect to data-names, PICTURE character-strings, and positions. There are two exceptions to this rule:

- a. The FILLER data items at the end of each of the above standard label formats may be replaced by descriptions of additional data items to be included in the label records. These additional data items, if present, may be processed in USE procedures.
- b. The VALUE clauses within the above label record descriptions do not result in automatic moves of the specified literals to output labels. All standard items within Series 60/6000 standard labels are automatically processed by the input-output system.

Example:

```
SPECIAL-NAMES.  
  GTIME IS TIME-VALUE.  
  GIN IS CONTROL-INPUT.  
.  
FILE-CONTROL.  
  SELECT PAYROLL-ACTIONS ASSIGN TO A1 FOR MULTIPLE REEL.  
  SELECT NEW-PAY-MASTER ASSIGN TO N1 FOR MULTIPLE REEL.  
.  
FD PAYROLL-ACTIONS  
  DATA RECORD IS ACTION-ITEM  
  LABEL RECORDS ARE STANDARD  
  VALUE OF ID IS ACTION-IDENT.  
01 ACTION-ITEM ...  
.  
FD NEW-PAY-MASTER  
  DATA RECORD IS NEW-MASTER-RECORD  
  LABEL RECORDS ARE STANDARD.  
01 NEW-MASTER-RECORD ...  
.  
WORKING-STORAGE SECTION.  
01 ACTION-IDENT.  
  02 FILLER PIC X(6) VALUE "PAYACT".  
  02 ACTION-DATE PIC 9(6).  
01 NEW-MASTER-IDENT.  
  02 FILLER PIC X(6) VALUE "PAYMST".  
  02 NEW-DATE PIC 9(6).  
01 RUN-TIME.  
  02 RUN-DATE PIC 9(6).  
  02 RUN-CLOCK PIC 9(6) USAGE COMP-3.
```

Example (cont):

```
PROCEDURE DIVISION.  
DECLARATIVES.  
NEW-MASTER-FILE-HEADER SECTION.  
    USE AFTER BEGINNING LABEL ON NEW-PAY-MASTER.  
    .  
    .  
    MOVE NEW-MASTER-IDENT TO IDENTIFICATION.  
    MOVE 32 TO RETENTION-PERIOD.  
    .  
    .  
END DECLARATIVES.  
    .  
    .  
    ACCEPT ACTION-DATE FROM CONTROL-INPUT.  
    OPEN INPUT PAYROLL-ACTIONS.  
    .  
    .  
    ACCEPT RUN-TIME FROM TIME-VALUE.  
    MOVE RUN-DATE TO NEW-DATE.  
    OPEN OUTPUT NEW-PAY-MASTER.  
    .  
    .
```

If the system input file (GIN) contained a record with the six-digit date in columns 1 through 6, each header label on the PAYROLL-ACTIONS file would be checked for a file identification (title) of the form "PAYACTmmddyy".

Each header label of the NEW-PAY-MASTER file will have a file identification of the form "PAYMSTmmddyy" and a retention period of 32 days.

LABEL-NAME OPTION

A label-name in the LABEL RECORD(S) clause must be one of the set of four fixed label-names associated with the Series 60/6000 standard label format specifications. (Refer to the STANDARD option.)

All Procedure Division references to label-name-1, or to any items subordinate to label-name-1, must appear within USE procedures.

If more than one label-name is specified, it is an implicit redefinition of the same label area.

The fixed label-names and their associated Series 60/6000 standard label records are described below:

- a. BEGINNING-FILE-LABEL: Appears only once for a file, preceding the first data record of the file, and contains information about the file.
- b. BEGINNING-TAPE-LABEL: Appears at the physical beginning of each reel, with the exception of the first reel on the file, preceding all other information, and contains information about the reel.
- c. ENDING-FILE-LABEL: Appears only once for a file, following the last data record on the last reel of a file, and contains information about the file.

- d. ENDING-TAPE-LABEL: Appears at the physical end of a reel, with the exception of the last reel of the file, following the last data record, and contains information about the reel.

Since these fixed label-names are associated with the Series 60/6000 standard label descriptions, their formats are inferred by the compiler and it is not necessary to define them within the file description. Omitting the label record descriptions within this application is operationally equivalent to using the STANDARD option.

However, user-defined label record descriptions may be associated with these four fixed label-names and the compiler will use these descriptions in the source program. The input-output system will recognize only four types of label processing; BEFORE BEGINNING LABEL, AFTER BEGINNING LABEL, BEFORE ENDING LABEL, and AFTER ENDING LABEL. Due to this restriction, it is the responsibility of the user to determine whether the file being processed is a reel label or a file label.

Example:

```
FILE-CONTROL.
  SELECT TRANSIT-FILE ASSIGN TO TR FOR MULTIPLE REEL.
  .
FD TRANSIT-FILE
  DATA RECORDS ARE TRANSIT-A, TRANSIT-B
  LABEL RECORDS ARE
    BEGINNING-FILE-LABEL
    BEGINNING-TAPE-LABEL
    ENDING-FILE-LABEL
    ENDING-TAPE-LABEL.
01 ENDING-FILE-LABEL.
  02 SENTINEL PIC X(6).
  02 BLOCK-COUNT PIC 9(6).
  02 REEL-PROOF-TOTAL PIC 9(16)V99.
  02 FILE-PROOF-TOTAL PIC 9(16)V99.
  02 FILLER PIC X(36).
01 ENDING-TAPE-LABEL.
  02 SENTINEL PIC X(6).
  02 BLOCK-COUNT PIC 9(6).
  02 REEL-PROOF-TOTAL PIC 9(16)V99.
  02 FILLER PIC X(54).
  .
PROCEDURE DIVISION.
DECLARATIVES.
FILE-TRAILER SECTION.
  USE AFTER ENDING LABEL ON TRANSIT-FILE.
  IF SENTINEL EQUALS "EOF" MOVE
    WORKING-REEL-PROOF-TOTAL TO REEL-PROOF-TOTAL
    ADD WORKING-REEL-PROOF-TOTAL TO WORKING-FILE-PROOF-TOTAL
    MOVE WORKING-FILE-PROOF-TOTAL TO FILE-PROOF-TOTAL
    ELSE MOVE WORKING-REEL-PROOF-TOTAL TO REEL-PROOF-TOTAL
    ADD WORKING-REEL-PROOF-TOTAL TO WORKING-FILE-PROOF-TOTAL
    MOVE 0 TO WORKING-REEL-PROOF-TOTAL.
END DECLARATIVES.
  .
  .
```

The above example describes a file that utilizes some of the unused space in the Series 60/6000 standard labels. Since the input-output system is unable to differentiate among the label formats, the VALUE OF clause should not be used to set the IDENTIFICATION field.

RECORD CONTAINS Clause

This optional clause may be used for documentation. The size of each record type is determined from information in its record description entries, and is not affected by this clause. Integer-1 is the number of characters in the smallest data record, and integer-2 is the number of characters in the largest data record. If the data records in the file are of uniform size, integer-1 is omitted and integer-2 is the exact number of characters in each record.

The size is specified in terms of the number of standard characters contained within the logical record, regardless of the actual method used to represent the items within the logical record. The sizes of the records are established according to the rules for determining the size of a group item.

RECORDING MODE Clause

The RECORDING MODE clause is used to specify the format or organization of data on magnetic tape.

Whenever an APPLY SYSTEM STANDARD phrase is specified in the I-O-CONTROL paragraph for a file, the RECORDING MODE clause, if one is given for the file, must specify BINARY HIGH DENSITY.

If any data item or report item associated with a file has any usage other than USAGE DISPLAY, the BCD option must not be used.

It is not necessary to specify the BCD and LOW DENSITY options, except for magnetic tape files, and these options are recommended only for compatibility on magnetic tape. If the actual peripheral device used is a printer, a card reader, or a punch, the input-output subroutines in the object program automatically use the density appropriate to the device. If a file is intended for cards or printer via media conversion (either input or output), the recording mode utilized should be binary high density.

If the RECORDING MODE clause is omitted, the file will be assumed to be recorded in binary high density mode.

In the computer, all data values are represented either by binary numbers or by binary-coded internal 'characters'. For example, the letter A is represented internally by the six-bit binary code 010001. Use of the term 'binary' in describing the recording mode reflects the fact that in that mode the peripheral medium represents the data with exactly the same binary bit configuration as in memory. Any data which can be stored in memory can be written to and retrieved from a peripheral in the same configuration via binary mode. This is not true of BCD mode. BCD mode on magnetic tape permits only binary-coded characters, not binary numbers. For example, the character with internal binary code 001010, whose graphic is '[' (left bracket), cannot be represented in BCD mode on magnetic tape.

Series 60/6000 computers provide BCD mode for compatibility with other computers. Of the 64 possible binary-coded characters, a certain subset has wide usage for data processing applications in many machines. Included are the letters, the space character, the digits, and certain editing characters. These characters are represented by different binary codes on various machines, but their representation on BCD magnetic tape is standardized. For example, a space or blank has binary code 110000 in some machines, but has binary code 010000 in Series 60/6000 computers. If a space character is written to magnetic tape in BCD mode on such a 'stranger' machine, it will be read as a space character by the Series 60/6000 computer.

Space character in 'stranger' computer	Space character on BCD tape	Space character in Series 60/6000
110000	→ 010000	→ 010000

If the recording mode is binary, each record in the file may contain an arbitrary mixture of binary (COMPUTATIONAL or COMPUTATIONAL-n) and BCD or character-oriented data. Only BCD information may appear in the BCD mode.

In Series 60/6000 COBOL, the following language elements require the binary recording mode on the file affected:

- ACCESS MODE SEQUENTIAL/RANDOM
- Block serial numbers
- FOR CARDS or FOR LISTING
- OCCURS...DEPENDING
- REPORT(S)
- System Standard Format
- USAGE COMPUTATIONAL or COMPUTATIONAL-n
- USAGE DISPLAY-2
- VLR format

All files on a multiple file tape must have the same recording mode.

Label records have the same recording mode as data records.

REPORT(S) Clause

The optional REPORT(S) clause specifies which report(s), described in the Report Section of the Data Division, belong to this file.

Each data-name entered in this clause must be a report-name specified in an RD entry in the Report Section. A report cannot be assigned to more than one file.

Multiple data-names listed in the REPORT(S) clause indicate that the file contains more than one report. The order in which the report-names are listed is not significant. If a file contains multiple reports which are not produced sequentially, then each report must be assigned a unique report code using the CODE clause in the RD entry. Each report line on the output device will then be labeled with the appropriate report code so that lines from the various reports can be easily distinguished.

At program execution, a file containing reports should be assigned to SYSOUT (system output) or to a high-speed peripheral device, normally magnetic tape, to be printed later via media conversion.

VALUE OF Clause

The VALUE OF clause in a file description entry is used to particularize the description of a data item in the label records associated with a file.

The referenced data-name should be qualified when necessary; however, the data-names cannot be subscripted, indexed, or described with the USAGE IS INDEX clause. The data-names which will be checked or modified in the label processing must be contained in one of the label records.

If the file is opened as input, the appropriate label routine verifies that the value of the label record data item is equal to the value of the specified literal.

If the file is opened as output, the value of the label record data item is made equal to the value of the specified literal at the appropriate time.

Data-name-1 and data-name-3 can only be the fixed label item names IDENTIFICATION and RETENTION-PERIOD. Either or both names may be given.

If IDENTIFICATION is specified, the action in the object program depends upon the use of the file. For an input file, the object program's label check routine verifies that the value of the IDENTIFICATION in beginning labels of the file is equal to the given literal. For an output file, the literal is implicitly moved to IDENTIFICATION before a beginning label is written. The literal associated with IDENTIFICATION must be a nonnumeric literal of no more than 12 characters.

For mass storage files, the VALUE OF IDENTIFICATION clause is ignored since a label record is not present on the external device. However, the standard label USE procedures are engaged at OPEN and CLOSE for such files.

RETENTION-PERIOD has no significance for input files. For an output file, the given literal is implicitly moved to RETENTION-PERIOD before a beginning label is written. The literal associated with RETENTION-PERIOD must be a positive integer not exceeding 999. The value 999 signifies permanent retention.

If a magnetic tape to be used for output has a RETENTION-PERIOD given in its beginning label, the object program's output routine checks that value against the current date to assure that the RETENTION-PERIOD has expired before it begins writing new data on the tape.

If the label records are user-defined, the referenced label record data items must be described in the same position within each label record associated with the file. It is not required that the data-names be the same in all of the records.

For additional information, refer to the description of the LABEL RECORD(S) clause in this section.

SECTION IV

RECORD DESCRIPTIONS

ELEMENTARY ITEM DESCRIPTION ENTRIES

Elementary items are data items that are not further subdivided. Refer to the COBOL Reference Manual for specific elementary item record formats. The following rules must be observed:

1. Each elementary entry must contain a PICTURE clause except for index data items (USAGE INDEX).
2. Level-number must be an integer in the range 1-49 or 77.
3. Data-name-1 or FILLER must immediately follow level-number.
4. If the REDEFINES clause is specified, it must be written immediately after data-name-1. The other clauses may be written in any order.
5. Only one of the two possible OCCURS clause formats can be present in a data description entry.

BLANK WHEN ZERO Clause

The optional BLANK WHEN ZERO clause is used in a data description entry to enable the blanking of an item when that item's value is zero.

This clause may be used only for an elementary item whose PICTURE is specified as numeric edited. It may not be used for variable-length items.

When this clause is used, the item contains only spaces when the value of the item is zero. When used for an item whose PICTURE is numeric, the category of the item is considered to be numeric edited.

Condition-Name Entry

Format 5 of the data description skeleton is used for each condition-name (refer to the COBOL Reference Manual). Each condition-name requires a separate entry with level-number 88. The condition-name entry specifies the name of the condition and the value, values, or range of values associated with the condition-name.

Condition-names are associated with a group or elementary item. The 'condition' is the truth or falsity of the proposition that the value of the item equals the value (or one of the values) associated with the condition-name, or falls within the specified range of values.

If condition-names are associated with a data item, the item is called a conditional variable. The condition-name entries for a particular conditional variable must follow the entry describing the data item with which the condition-name is associated. A condition-name can be associated with any data description entry that contains a level-number except for the following:

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

The following example illustrates a method in which a condition-name entry can be written:

```
03 GRADE PIC 9(2).  
  
88 PRIMARY VALUE IS 1.  
88 SECOND VALUE IS 2.  
.  
.  
88 GRADE-SCHOOL VALUES ARE 1 THRU 6.  
88 JUNIOR-HIGH VALUES ARE 7 THRU 9.  
88 HIGH-SCHOOL VALUES ARE 10 THRU 12.  
88 GRADE-ERROR VALUES ARE 0, AND 13 THRU 99.
```

The application of condition-names in the Procedure Division is described in Section XII of this manual.

Condition-name entries may appear only in the File Section or Working-Storage Section of the Data Division.

The condition-names described in the Data Division are functionally equivalent to those associated with the ON or OFF status of switches in the SPECIAL-NAMES paragraph of the Environment Division. The latter type of condition-name is not defined in a condition-name entry.

COPY Clause

The COPY clause is used in association with a file-name or a data-name in FD, SD, RD, data description, and report group description entries. The COPY clause is used to direct the compiler to duplicate text from the source program or a library into the source program.

COBOL provides two distinct and mutually exclusive COPY functions. The first function, referred to as HIS COPY, has been available in the pre-standard version of COBOL. The second function, referred to as the American National Standard COPY, represents Level 2 of the American National Standard COBOL X3.23-1968 library facility.

Section VIII of the COBOL Reference Manual contains the descriptions, formats, and syntax rules for the COPY functions. Section XIV of this manual describes the COBOL library facility with associated source library formats and related examples of the COPY options.

JUSTIFIED Clause

The JUSTIFIED clause in a data description entry is used to specify nonstandard positioning of data within a receiving data item.

When the storage area of a data item receives data resulting from an arithmetic or data movement procedure, the standard rules for positioning the value within the receiving area are:

1. If the receiving item is numeric, the assumed decimal point of the sending data item is aligned with that of the receiving data item, and each end of the value is zero-filled or truncated, as required.
2. If the receiving item is an edited numeric item with an actual or implied decimal point, the value is aligned by decimal point with fill or truncation on either end, as required.
3. If the receiving item is alphanumeric (other than a numeric edited data item) or alphabetic, the sending data item 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.

The JUSTIFIED clause can be used to reverse the standard rule for nonnumeric items (rule 3 above). This causes the value to be right-justified with space-fill or truncation on the left.

The JUSTIFIED clause can be specified only at the elementary item level. It must not be specified for an item that has any of the following properties:

1. Class numeric or numeric edited.
2. Usage other than DISPLAY.
3. Actual or assumed decimal point.

Level-Number/Data-Name Entries

Level-number entries show the hierarchy of data within a logical record or report group. They are also used to identify entries for condition-names, working-storage data items, noncontiguous data items, and the RENAME clause.

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

Data description entries that are subordinate to FD or SD entries may use level-numbers with values in the range 01 (or 1) through 49, or level-number 66, or level-number 88. Report group description entries that are subordinate to an RD entry may only use level-numbers whose values are in the range 01 through 49.

Multiple level 01 entries that are subordinate to a level indicator (FD, SD) represent implicit redefinitions of the same area. The first occurrence defines an area and all subsequent occurrences redefine the same area.

A level 01 entry must be used to identify the first entry in each record description and in each report group description. When no real concept of level exists, the special level-numbers 66, 77, and 88 are assigned as follows:

1. Level-number 66 is used in a data description entry to identify RENAMEs entries. Format 4 of the data description skeleton described in the COBOL Reference Manual must be used.
2. Level-number 77 entries may be used in the Working-Storage Section to identify noncontiguous data items. The level-number 77 entry must be specified as shown in Format 3 of the data description skeleton.
3. Level-number 88 entries are used to define condition-names associated with conditional variables and may be used in the Working-Storage Section and the File Section. Format 5 of the data description skeleton must be used.

OCCURS Clause

The OCCURS clause in a data description entry is used to define tables of repeated items. It is required when the data either might not exist or might occur more than once. When the OCCURS clause is not specified, one occurrence is assumed. This clause must not appear in entries with level-numbers 01, 66, 77, or 88, nor with entries which have subordinate variable occurrence data items.

If an item is described with the OCCURS clause, its data-name must be subscripted or indexed in all references. If it is a group item, then each data-name belonging to the group must be subscripted or indexed in each reference except for the SEARCH statement. If an item's record description entry includes an OCCURS clause, the other descriptive clauses apply to each occurrence of the item being described.

A table item may be a conditional variable. The condition-name entries follow the conditional variable, as usual, and do not contain OCCURS clauses. Any references to such condition-names require subscripts or indexing.

The INDEXED phrase is required when the subject of this data description entry (or an entry subordinate to this entry if it is a group item) is to be referred to by indexing. The index-names identified in the OCCURS clause are not defined elsewhere since the allocation and format of these index-names is hardware dependent. Not being data, these index-names cannot be associated with any data hierarchy. The index-names must be unique within the program.

The KEY phrase is used to indicate that the repeated data is arranged in ascending or descending order according to the values contained in the data-names referenced by the phrase. The data-names are listed in their descending order of significance, from most significant to least significant.

If the data-name in the KEY phrase is not the subject of this entry, then:

- a. All of the items identified by the data-names used in the KEY phrase must be contained in the group item that is the subject of this entry.
- b. The items identified by the data-names in the KEY phrase may not be described by an entry that contains an OCCURS clause or be subordinate to an entry containing an OCCURS clause.

If the number of occurrences may vary, Format 2 must be specified. Integer-1 indicates the minimum number of occurrences and integer-2 specifies the maximum number of occurrences. Integer-1 may be zero to indicate that the data might not exist. In any case, integer-2 must not be a zero.

The DEPENDING phrase in Format 2 is required only when the end of the occurrences of the item cannot otherwise be determined.

A group or record is said to have 'variable length' if any item subordinate to it is described with Format 2. The following restrictions apply to any item with a variable number of occurrences and to any group containing such an item:

1. It cannot be subordinate to an OCCURS item.
2. It cannot be redefined or be subordinate to an item which is redefined.
3. It cannot appear in a redefinition.
4. It must not be used for partitioned records.

Unless the DEPENDING phrase is also specified, the integer-1 option does not affect the object program. (The DEPENDING phrase must not be specified unless integer-1 is also specified.)

The results obtained from the use of OCCURS...DEPENDING are generally different from one computer to another.

The DEPENDING phrase may not be specified in the record descriptions of a sort file or a merge file.

In its internal format, an OCCURS...DEPENDING table has a memory area of sufficient size to hold the maximum number of occurrences. The significant items are understood to appear in successive positions at the beginning of the table; unused item positions at the end of the table are called table residue. The contents of the residue area are unpredictable. The table residue is suppressed on the peripheral device.

In the File Section, OCCURS...DEPENDING results in suppressing table residue on the peripheral device, as described below, when the following criteria are met:

1. Data-name-1 is described as COMP-1.
2. Data-name-1 is subordinate to the same record description entry.
3. Data-name-1 precedes the data description entry containing the OCCURS...DEPENDING clause.

When compression is to take place, the compiler will automatically generate a process area for the file, regardless of whether or not the APPLY PROCESS AREA phrase is specified in the I-O-CONTROL paragraph.

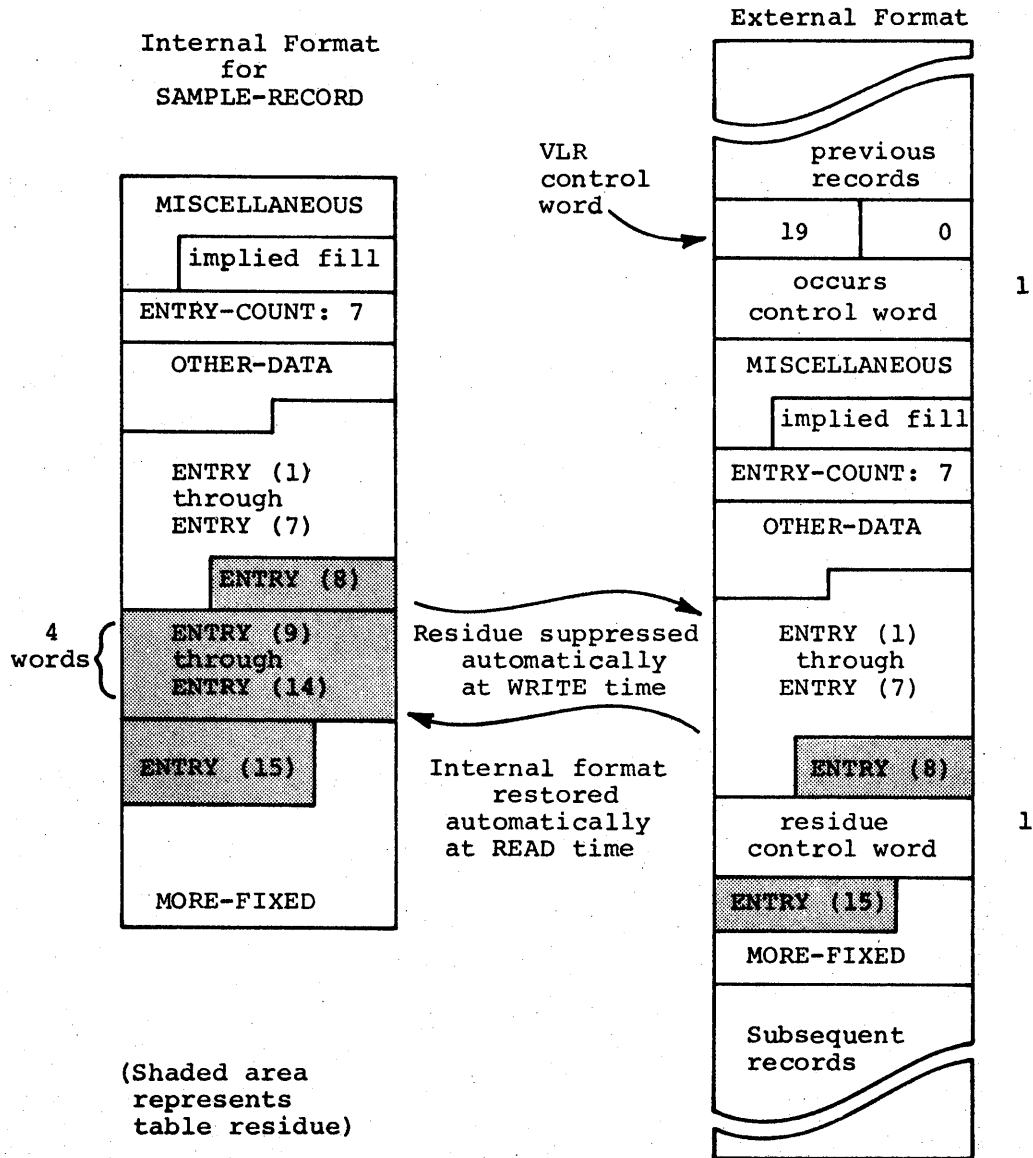
When a WRITE statement references a record that may be compressed, the object program examines the output record for opportunities for residue suppression. Such opportunities are rejected unless two or more machine words can be suppressed. In the latter case, suppression proceeds on a machine word basis; the whole-word portion of the residue of each table is replaced by a single control word. Each variable-length table in the record presents an opportunity for residue suppression. Actual suppression takes place in an implicit move from the process area to the output buffer.

When a READ statement references a file containing records that may be compressed, the record is implicitly moved from the input buffer to the process area and expanded to the format it had in memory prior to residue suppression.

Consider a data record described as follows:

- 01 SAMPLE-RECORD.
 - 02 FIXED-PORTION.
 - 03 MISCELLANEOUS; PICTURE X(20).
 - 03 ENTRY-COUNT; PIC 99 COMPUTATIONAL-1.
 - 03 OTHER-DATA; PICTURE X(16).
 - 02 VARIABLE-LENGTH-TABLE.
 - 03 ENTRY; PICTURE X(4); OCCURS 1 TO 15 TIMES DEPENDING ON ENTRY-COUNT.
 - 02 MORE-FIXED; PICTURE X(20).

The following illustration shows the relationship of the internal and external formats of SAMPLE-RECORD when the value of ENTRY-COUNT happens to be seven:



¹The occurs and residue control words contain a word count in the lower half and a relative beginning word position in the upper half.

When the OCCURS...DEPENDING clause is specified for any record of a file that may be compressed, the data format of the file is affected as follows:

1. The variable-length record (VLR) format is automatically applied.
2. The recording mode must be binary (explicitly or implicitly).
3. In addition to any residue suppression control words needed, each record on the peripheral device begins with a control word required for reconstruction.

In the DEPENDING phrase, data-name may be qualified but must not be subscripted.

To enhance object program efficiency, table items should be referenced using indexes or referenced using subscripts described with USAGE COMP-1. Erroneous data will be obtained if the subscript value is greater than zero but less than the minimum occurrence number specified in the OCCURS clause. In the EIS mode, subscripts must contain a value greater than zero; otherwise, an F0 memory address fault will occur.

If the OCCURS clause is used, there is no implication that the data item is synchronized. To obtain improved object program efficiency, consideration should be given to specifying the SYNCHRONIZED clause for table items.

PICTURE Clause

The function of the PICTURE clause is to represent the standard data format of an elementary item, to describe the general characteristics of the item, and to enable special report editing.

A PICTURE clause may be specified only at the elementary item level.

The PICTURE clause must be specified for every elementary data item except an index data item, in which case the use of this clause is prohibited.

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

The maximum number of symbols allowed in a PICTURE character-string is 30.

If a PICTURE clause specifies insertion editing, part of the receiving item is subject to a size limitation. The limited part of the receiving item consists of that portion of the item bounded on the left by the leftmost noninsertion character and bounded on the right by the rightmost noninsertion character.

Example:

<u>PICTURE</u>	<u>Limited Part</u>
BBB999,999.99000	999,999.99
B(n)X(n)B(n)X(n)B(n)	X(n)B(n)X(n)
X(n)B(n)9(n)	X(n)B(n)9(n)

The maximum size of the limited part of the receiving item may be determined from the following relationship:

$$108 \geq 3e+n$$

where:

e represents the number of simple and/or special insertion symbols ('.', 'B', 'O', '.') exclusive of leading and trailing insertion symbols.

n represents the number of noninsertion symbols ('X', '9', 'A').

Example:

<u>PICTURE</u>	<u>e</u>	<u>n</u>	<u>3e+n</u>	<u>Limit Exceeded?</u>
XB(35)XX	35	3	108	No
B(5)XB(35)XXB(5)	35	3	108	No
X(55)BX(50)	1	105	108	No
B(5)X(55)BX(50)B(5)	1	105	108	No
XB(10)XB(25)X	35	3	108	No
XB(35)X	35	2	107	No
XB(36)X	36	2	110	Yes
XB(10)XB(20)XB(5)X	35	4	109	Yes
B(10)XB(20)XB(5)XXB(10)	25	4	79	No

If the maximum size of the limited part of the receiving item is exceeded, the compiler will issue the following error message:

```
ITEM IS TOO LARGE FOR EMBEDDED INSERTION
CHARACTERS-TRUNCATED BUT ALSO REGARDED AS UNUSABLE.
```

The allowable characters (or symbols) which may appear in a character-string are defined and described in the COBOL Reference Manual according to the category definition of the data item being described by the PICTURE clause and according to the associated MOVE category of the data item. Five categories of data may be described with a PICTURE clause; alphabetic data items, numeric data items, alphanumeric data items, alphanumeric edited data items, and numeric edited data items.

EDITING RULES

Two general methods are used to perform editing in the PICTURE clause, either by insertion or by suppression and replacement. The four types of insertion editing are:

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

The two types of suppression and replacement editing are:

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

The type of editing which may be performed upon an item depends on the category to which the item belongs. The following list indicates which type of editing may be performed upon a given category:

<u>Category</u>	<u>Type of Editing</u>
Alphabetic	None
Numeric	None
Alphanumeric	None
Alphanumeric Edited	Simple insertion, '0' and 'B'
Numeric Edited	All, subject to the floating insertion rule given below
Any Variable-Length Item	None

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

Refer to the PICTURE clause in the COBOL Reference Manual for detailed descriptions of the six types of editing listed above and to determine the order of precedence when using characters as symbols in a PICTURE character-string.

REDEFINES Clause

The REDEFINES clause is used to apply alternative descriptions to the same memory area. When it appears in a record description entry, the REDEFINES clause must immediately follow data-name-1 (the subject data-name). The level-numbers of data-name-1 and data-name-2 must be identical but must not be level-number 66 or level-number 88.

The REDEFINES clause must not be specified in level 01 entries in the File Section. The same memory area is implicitly redefined by each successive level 01 entry belonging to the file.

Data-name-2 is the data-name of the item which occupies the memory area being redefined. Redefinition begins at data-name-2 and continues until a level-number less than or equal to that of data-name-2 is encountered. The record description entries giving the new descriptions of the memory area must immediately follow the entries giving the prior definition.

Example:

```
02 A ...
   03 ...
   03 ...
02 B REDEFINES A ...
```

If several alternative descriptions are to be applied to a memory area, each new REDEFINES clause must refer to the data-name of the entry that originally defined the memory area.

Continuing the above example:

```
02 B REDEFINES A ...
02 C REDEFINES A ...
```

The level-number of data-name-1 must be the same as that of data-name-2. The area being redefined is that described by the data-name-2 entry and all of its subordinate entries, terminated by the data-name-1 entry.

The following restrictions must be observed:

1. The data description entry for data-name-2 cannot contain an OCCURS clause nor can data-name-2 be subordinate to an entry containing an OCCURS clause. Neither the original definition nor any subsequent redefinitions of the area can include an item whose size is variable as defined for the OCCURS clause.
2. The VALUE clause must not appear in the record description entry for data-name-1, nor in any of its subordinate entries, except in condition-name entries.
3. If a numeric data item is redefined by an alphanumeric data item or vice versa in the EIS mode, the numeric item must contain only numeric data when it is referenced in an arithmetic statement. A reference to a numeric item that contains nonnumeric data will result in the abort message ILLEGAL EIS DATA at execution time.

4. When the REDEFINES clause is specified in the Working-Storage Section and more than fifty noncontiguous data items (level 77) are defined, the REDEFINES clause and the item it redefines must be included in the same group of fifty items; that is, in the first fifty level 77 items, or the second fifty level 77 items, etc.
5. Redefinition of an alphabetic or alphanumeric data item by a double-precision numeric COMPUTATIONAL, COMPUTATIONAL-1, or COMPUTATIONAL-2 data item may result in an incorrect (odd) address for the numeric item and the following error message:

ER REDEFINITION EXCEEDS CAPACITY OF ORIGINAL DEFINITION

Example:

```

77 DBLP PIC 9(18) COMP-1.
77 LWORD PIC X(6).
77 DBLWD PIC X(12).
77 D918CMP1 REDEFINES DBLWD PIC 9(18) COMP-1.

```

In the above example, an odd address would be assigned to D918CMP1. If the redefined and the redefining item entries are reversed in the above example, a correct address will be assigned as shown below:

```

77 D918CMP1 PIC 9(18) COMP-1.
77 DBLWD REDEFINES D918CMP1 PIC X(12).

```

The following combinations are permitted:

1. The record description entry for data-name-2 or entries subordinate to it may contain the VALUE clause without violating the above rules. (If an initial value is to be specified, it must be given in the first definition of the memory area.)
2. The record description entry for data-name-1 or entries subordinate to it may contain the OCCURS clause without violating the above rules. Furthermore, entries subordinate to data-name-1 may contain the OCCURS clause (an item containing a table can be redefined, or a redefinition can contain a table, but an item within a table cannot be redefined).

When initial values are to be specified for the items in a working-storage table, a combination of the VALUE, REDEFINES, and OCCURS clauses is often used.

Example:

```

02 TAX-RATES.
03 FILLER PICTURE V99 VALUE .14.
03 FILLER PICTURE V99 VALUE .15.
03 FILLER PICTURE V99 VALUE .16.
03 FILLER PICTURE V99 VALUE .17.
.
.   (21 additional entries)
.
02 RATE-TABLE REDEFINES TAX-RATES.
03 RATE PICTURE V99 OCCURS 25 TIMES.

```

The REDEFINES clause specifies only the redefinition of a memory area, not of the data items occupying the area. If the SYNCHRONIZED clause resulted in unused character positions in the original definition of the area, other than to the left of the first item being redefined, the new definition must account for all such character positions. If the first item in the original definition is synchronized, unused character positions to its left are not considered a part of the area being redefined.

Noncontiguous data items are automatically synchronized to produce more efficient coding. If a noncontiguous data item is redefined, the redefining data description entry will be assigned the same synchronization as the redefined data description entry.

RENAMES Clause

The RENAMES clause is a level-number 66 data description entry that may be used to supply an alternative data-name for an elementary item, or to indicate regrouping of items. One or more RENAMES entries may be applied to the data items within a logical record. They must be specified as the last entries subordinate to the level 01 entry, and cannot themselves have subordinate entries.

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

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

If data-name-3 is specified, data-name-1 is a group item that includes all the elementary items starting with data-name-2 (if data-name-2 is an elementary item) or starting with the first elementary item in data-name-2 (if data-name-2 is a group item), and concluding with data-name-3 (if data-name-3 is an elementary item) or concluding with the last elementary item in data-name-3 (if data-name-3 is a group item). The data description entry for data-name-2 must precede that of data-name-3. If data-name-2 is a group item, data-name-3 must not belong to that group.

If data-name-3 is not specified, data-name-1 assumes the properties of data-name-2; its group or elementary status is thereby determined, and if data-name-2 is an elementary item, then data-name-1 assumes the same description.

SYNCHRONIZED Clause

The optional SYNCHRONIZED clause in a data description entry is used to specify the alignment of an elementary item within a computer word or words. It may appear only in an elementary entry.

The SYNCHRONIZED clause indicates that the item's storage area is organized in integral computer word lengths. The SYNCHRONIZED clause depends upon the particular characteristics of a computer. In Series 60/6000 COBOL, SYNCHRONIZED relates the item's position to the six-character word length of the machine. See Section II.

A synchronized data item utilizes the smallest number of consecutive whole words that can contain it. Its position in the words it occupies is independent of the preceding and following record description entries.

The SYNCHRONIZED LEFT option causes the data item to be oriented in such a way that its first character occupies the first position of the initial word. Fill characters follow the last data character of the item in the unused positions of the last word, as the following example illustrates:

<u>Description of Item</u>	<u>Resulting Orientation</u>
03 D PIC X(2) SYNCHRONIZED LEFT.	D D (fill)
03 E PIC X(6) SYNCHRONIZED LEFT.	E E E E E E
03 F PIC X(7) SYNCHRONIZED LEFT.	F F F F F F F (fill)

The SYNCHRONIZED RIGHT option causes the data item to be oriented in such a way that its last character occupies the last position of the final word. Fill characters precede the first data character of the item in the unused positions of the first word, as the following example illustrates:

<u>Description of Item</u>	<u>Resulting Orientation</u>
03 A PIC X(2) SYNCHRONIZED RIGHT.	(fill) A A
03 B PIC X(6) SYNCHRONIZED RIGHT.	B B B B B B
03 C PIC X(7) SYNCHRONIZED RIGHT.	(fill) C C C C C C C

A reference to any group item, regardless of its level-number, does not include the unused character positions of the first elementary item in the group if the group item is synchronized right.

Since synchronized items never share the words they occupy with other items, fill characters may be required in the word preceding the first word of a synchronized item, as the following example illustrates:

<u>Description of Item</u>	<u>Resulting Orientation</u>
01 G.	
02 H PIC X(8).	H H H H H H
02 I PIC X.	H H I (fill)
02 J PIC X(3) SYNCHRONIZED RIGHT.	(fill)J J J
02 K PIC X(4).	K K K K L (one fill)
02 L PIC X.	M M M M M M
02 M PIC X(6) SYNCHRONIZED LEFT.	

The fill characters noted above are implied by the combination of the item's size and the SYNCHRONIZED clause. The fill character positions cannot be described (except with the REDEFINES clause). An attempt to specify such fill characters via FILLER will cause undesired results, as the following example illustrates:

<u>Description of Item</u>	<u>Resulting Orientation</u>
01 P.	
02 Q PIC X(8).	Q Q Q Q Q Q
02 R PIC X.	Q Q R F I L
02 FILLER PIC X(6).	L E R (fill)
02 S PIC X(3) SYNCHRONIZED RIGHT.	(fill) S S S

A special type of word synchronization is applied to COMPUTATIONAL-n items. Each COMPUTATIONAL-n item monopolizes the computer word or words that contain it, so that fill characters may be required to complete the word preceding a COMPUTATIONAL-n item. Unlike DISPLAY-n items, however, COMPUTATIONAL-n items fully occupy the word or words which contain them. Thus, fill characters will never be present within a word that is used as a COMPUTATIONAL-n item.

The size of a synchronized data item excludes unused (fill) character positions.

COMPUTATIONAL-n items occupying one word are:

1. COMPUTATIONAL items in one of the following categories:

<u>Number of Integral Digits</u>	<u>Number of Fractional Digits</u>
1-8	0
0-5	1-3
0-3	4-5
0	6-8

2. COMPUTATIONAL-1 or COMPUTATIONAL-2 items with no more than eight digits specified in their data descriptions.
3. COMPUTATIONAL-3 items.

COMPUTATIONAL-n items occupying two words are:

1. COMPUTATIONAL items not in one of the above categories.
2. COMPUTATIONAL-1 or COMPUTATIONAL-2 items with more than eight digits specified in their data descriptions.

A one-word COMPUTATIONAL-n item may occupy any word of a record. A two-word COMPUTATIONAL-n item, however, must begin an even number of words from the beginning of the record which contains it. Fill characters may precede such a two-word item as the following example illustrates:

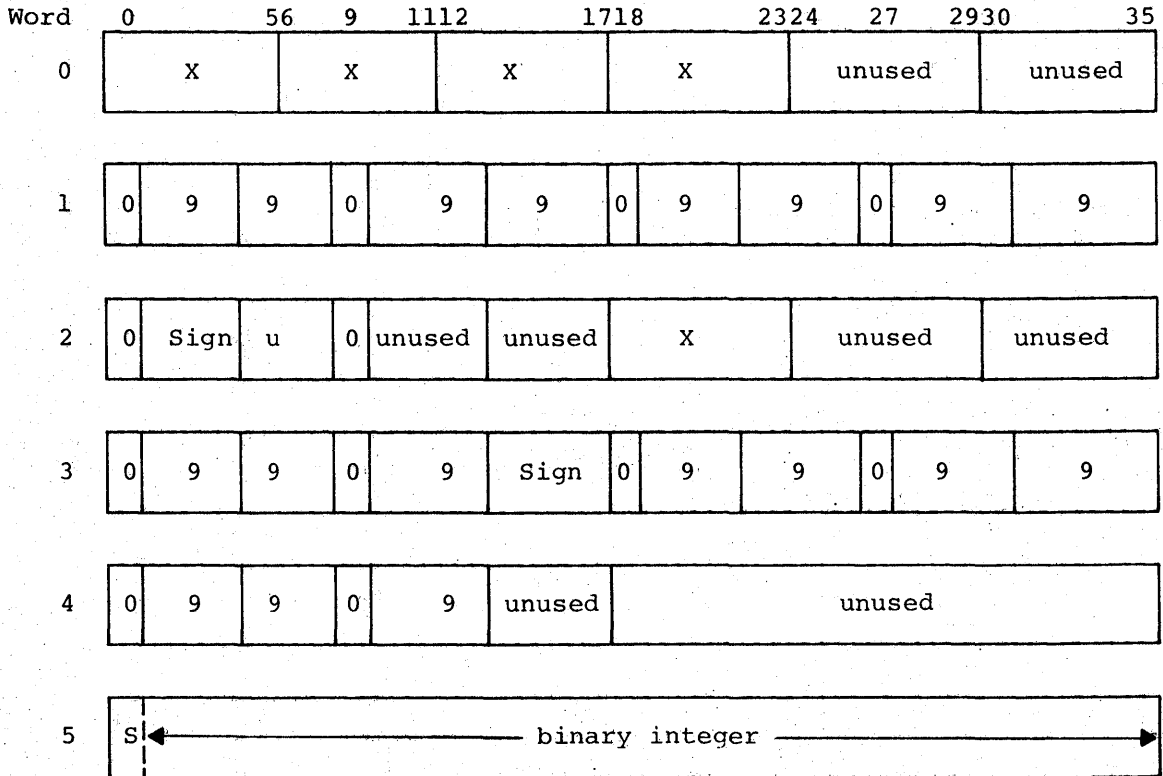
<u>Description of Item</u>	<u>Resulting Orientation</u>
01 A.	
02 B PICTURE 99V99 COMPUTATIONAL.	- - B B B B
02 C PICTURE 9(10)V99 COMPUTATIONAL.	- (fill) - C C C C C C C C C C

Packed decimal items are aligned on word or half-word boundaries whereas the other computational USAGE items are aligned on full-word boundaries. The following is an example of space allocation for a record description entry containing packed decimal data items:

- ```

01 REC-NAME.
02 FIELD1 PIC X(4).
02 PACK1 PIC S9(7)V9 USAGE COMP-4.
02 FIELD2 PIC X.
02 PACK2 PIC S9(3) COMP-4.
02 PACK3 PIC 9(7) COMP-4.
02 FIELD3 PIC S9(6) COMP-1.

```



To minimize the amount of leading fill resulting from synchronized items, such items should be grouped together within a record description, rather than scattering synchronized items among nonsynchronized items. Two-word COMPUTATIONAL-n items should be grouped together and all COMPUTATIONAL-n items should be grouped with synchronized items.

The fill characters resulting from the SYNCHRONIZED clause or COMPUTATIONAL-n usage appear in both the record's internal format and its external format. Otherwise, when the SYNCHRONIZED clause is omitted, data storage areas are assigned without regard to computer word length. The last character position of a given item is adjacent to the first character position of the next item. However, a level 01 or level 77 item always begins in a new word; it starts in the leftmost character position unless the SYNCHRONIZED RIGHT option is specified or the class of the item is numeric. In the latter cases, the data item is aligned to end in the rightmost character position in a word. A redefinition of the data item will be aligned to correspond with the synchronization of the level 01 or level 77 data item.

If a synchronized area is being redefined and the first item in the original definition is SYNCHRONIZED RIGHT, the area being redefined begins in the leftmost character position of the first word allocated to the original item. If the last item of the original definition is SYNCHRONIZED LEFT, the area being redefined extends to the rightmost character position of the last word allocated to the original item.

If the data description of an item contains the SYNCHRONIZED clause and an operational sign, the sign of the data item will appear in the least significant character position of the data item, regardless of whether the item is SYNCHRONIZED LEFT or SYNCHRONIZED RIGHT.

If an OCCURS data item is synchronized, each occurrence of the item will be synchronized.

#### USAGE Clause

The USAGE clause in a data description entry is used to indicate the dominant use of a data item or the manner in which a data item is to be represented in memory. It does not affect the use of the data item, although the rules for some statements in the Procedure Division may restrict the use of the USAGE clause for the operands referenced.

If the usage of a data item is not specified, it is assumed to be USAGE DISPLAY.

The USAGE clause may be specified at any level of a hierarchical structure. If this clause is specified at a group level, it applies to all of the subordinate elementary items in the group and no subordinate item may specify a different usage.

The external format of a data item (as it is stored on a peripheral device) and its internal format (as it is stored in memory) are always the same.



## DISPLAY ITEMS

When the USAGE clause specifies a display item, the data item is stored in the form of one or more standard data format characters. The USAGE clause permits a choice of the following internal formats for display data items.

### USAGE DISPLAY Items

USAGE DISPLAY represents character-oriented data. The data item is stored in the native (Series 60/6000) six-bit character set. The PICTURE clause description may imply alphabetic, alphanumeric, numeric, or numeric edited data items.

USAGE DISPLAY is preferred for data that is to be printed or punched.

### USAGE DISPLAY-1 Items

USAGE DISPLAY-1 represents edited floating-point data. The data item is stored in the native (Series 60/6000) six-bit character set. The class of the item is implicitly alphanumeric, and it is formatted as a Numeric Representation Set 3 character-string as specified in the American National Standard for the Representation of Numeric Values in Character Strings for Information Interchange (X3.42-1975).

The format of a DISPLAY-1 data item must be specified via the PICTURE clause using the following special format of the PICTURE character-string:

$$\left\{ \begin{array}{c} + \\ - \end{array} \right\} 9.9(n)E \quad \left\{ \begin{array}{c} + \\ - \end{array} \right\} 99$$

The edited floating-point construct, similar to the D- and E-conversions of the FORTRAN language, is useful only for very large or very small values which, in a normal DISPLAY format, would begin or end with a long string of zeros. In particular, computations involving COMPUTATIONAL-2 items (see below) may sometimes produce results for which the DISPLAY-1 format is needed.

### USAGE DISPLAY-2 Items

USAGE DISPLAY-2 represents character-oriented data. The data item is stored in a nonnative six-bit character set. The character set is the commercial collating character set described in Appendix D.

The PICTURE clause description must imply a class of alphabetic or alphanumeric for the data item.

Although a DISPLAY-2 data item may not be compared to an item having any other USAGE, it may be compared to literals and figurative constants.

DISPLAY-2 data items must be explicitly moved to USAGE DISPLAY items if they are to appear on punched cards, printer listings, or similar external media. This function cannot be accomplished with a REDEFINES clause.

The PICTURE of a DISPLAY-2 data item must not specify editing.

#### COMPUTATIONAL ITEMS

Computational data item formats are used to obtain both internal and external space-saving and performance advantages. The internal format invoked for a computational item must not conflict with the data characteristics specified for the item in the PICTURE clause.

Computational items are not stored in a manner related to the character position subdivisions of a computer word. The storage techniques are described in the preceding SYNCHRONIZED Clause discussion and in Section II.

Since a stored computational item has no character orientation, an attempt to manipulate it as if it were made up of characters is meaningless. Thus, the storage area may be redefined for some distinct purpose but not, for example, to give separate access to the integral and fractional parts of the computational item. For similar reasons, a group MOVE statement involving computational items should normally entail only sending and receiving groups with similar descriptions; a MOVE CORRESPONDING statement should be used otherwise. COBOL rules do not require adherence to the suggestions given in this paragraph, but the user must ensure that the application of a group MOVE statement or of a redefinition is legal.

The preferred format for all items used as subscripts or as objects of the DEPENDING phrase in the OCCURS clause is either COMPUTATIONAL-3 or COMPUTATIONAL-1 with a PICTURE containing eight or less digits.

In most commercial data processing applications, particularly where dollars and cents are involved, a high degree of decimal precision is expected. Since the ultimate accuracy for maximum length composite of operands (18 digits) in arithmetic statements may not be attainable due to floating-point hardware limitations, COMPUTATIONAL-2 should not be used where decimal precision is required. The special decimal-precision processing applied to computational items compensates for the floating-point hardware limitations in most cases.

The USAGE clause permits a choice of the following internal formats for computational data items.

#### USAGE COMPUTATIONAL Items

USAGE COMPUTATIONAL represents decimal-precision binary data. The data item is stored as a synchronized signed floating-point binary number. The PICTURE clause description must conform to the rules for numeric items. Computational is the preferred usage for items involved in calculations within a processor that does not contain the Extended Instruction Set (EIS).

## USAGE COMPUTATIONAL-1 Items

USAGE COMPUTATIONAL-1 represents binary integer data. The data item is stored as a synchronized signed fixed-point binary integer. The PICTURE clause description must conform to the rules for numeric items and the assumed decimal point must be immediately to the right of the rightmost digit position.

If the PICTURE clause specifies eight or less digits, the item is stored as a single-precision binary integer; otherwise, it is stored as a double-precision binary integer. It is the responsibility of the user to ensure that a double-precision binary integer begins in an even word storage location.

Although it is stored as a binary number, a COMPUTATIONAL-1 item's value is equal to the decimal value of the item because COMPUTATIONAL-1 items are restricted to integral values.

## USAGE COMPUTATIONAL-2 Items

USAGE COMPUTATIONAL-2 represents floating-point binary data. The data item is stored as a synchronized signed floating-point binary number. The PICTURE clause description must conform to the rules for numeric items.

If the PICTURE clause specifies eight or less digits, the item is stored as a single-precision floating-point number; otherwise, it is stored as a double-precision floating-point number. It is the responsibility of the user to ensure that a double-precision floating-point number begins in an even word storage location.

The COMPUTATIONAL-2 usage is especially effective for the operands in an elaborate formula. Should an operand value or an intermediate or final result exceed  $10^{18}$ , or be less than  $10^{-18}$ , only the floating-point binary format provides enough significance to yield meaningful results.

The mantissa of a COMPUTATIONAL-2 item is a pure binary fraction and consequently is not necessarily exactly equivalent to the item's decimal value. The equivalence may be sufficiently close, however, for practical purposes.

## USAGE COMPUTATIONAL-3 Items

USAGE COMPUTATIONAL-3 represents single-precision binary integer data. The data item is stored as a synchronized signed single-precision fixed-point binary integer. The PICTURE clause description must conform to the rules for numeric items and the assumed decimal point must be immediately to the right of the rightmost digit position. The PICTURE clause may specify at most ten digits.

## USAGE COMPUTATIONAL-4 Items

USAGE COMPUTATIONAL-4 represents packed decimal data. The data item is stored as a synchronized fixed-point packed decimal number. The PICTURE clause description must conform to the rules for numeric items. If the PICTURE character-string specifies an operational sign, the sign will be stored as a separate digit; that is, the data item will be one digit larger than the number of character positions described in the PICTURE character-string. If USAGE COMPUTATIONAL-4 is specified for a group item, the group item itself will be considered to be alphanumeric.

Data using this format can be processed only on a computer that has the Extended Instruction Set (EIS) capability.

## INDEX ITEMS

The USAGE INDEX clause can be written at any level. When it is used to describe an elementary item, the item is called an index data item. If the USAGE INDEX clause describes a group, the elementary items that make up the group are all index data items. An index data item is used to contain a value that corresponds to the occurrence number of a table element. The actual content of the index data item may depend upon the description of the table element. In any case, the method of representation of this value is single-precision binary integer.

An index data item must not be a conditional variable.

An index data item can be referenced directly only in a SEARCH statement, a SET statement, or in a relation condition. An index data item can be part of a group that is referenced in a MOVE statement or an input-output statement, in which case no conversion takes place.

The BLANK WHEN ZERO, JUSTIFIED, PICTURE, SYNCHRONIZED, and VALUE clauses must not be used to describe group items or elementary items described with the USAGE INDEX clause. If a group item is described with the USAGE INDEX clause, the elementary items in the group are all index data items. The group item itself is not an index data item and must not be used in SEARCH statements, SET statements, or in a relation condition.



## VALUE Clause

The VALUE clause in a data description entry may be used to define the initial value of working-storage data items, or the values associated with a condition-name. Rules governing its use differ with the respective sections of the Data Division:

1. In the File Section, the VALUE clause is meaningful only in condition-name entries. A VALUE clause in a record description entry in the File Section does not cause the item to assume the given value in an output record. Instead, the value must be moved into the output record via Procedure Division statements.
2. In the Working-Storage Section, the VALUE clause may be used in condition-name entries, and it may also be used to specify the initial value of any other data item. In the latter case, it causes the item to assume the specified value at the start of object program execution. If the VALUE clause is not used in an item's description, the initial value is undefined.
3. In the Report Section, the VALUE clause causes the report data item to assume the specified value each time its report group is presented. This clause may be used only at the elementary level in the Report Section.

A figurative constant may be substituted wherever a literal is specified in the formats of the VALUE clause.

The VALUE clause must be consistent with the other elements of the item's description. 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 to each literal in the VALUE clause:

1. If the category of the item is numeric, each literal must be a numeric literal. It is aligned according to the item's description and the rules given under the JUSTIFIED clause. The literal must not have a value that would require truncation of nonzero digits. A signed numeric literal must be associated only with a signed numeric (S9) PICTURE character-string.

If the data description entry of a numeric item also contains the BLANK WHEN ZERO clause, the literal must be a nonnumeric literal.

2. If the category of the item is alphabetic, alphanumeric, alphanumeric edited, or numeric edited, all literals must be nonnumeric literals. The literal is aligned according to the item's description and the rules given under the JUSTIFIED clause. The number of characters in the literal must not exceed the size of the item. If the item's description specifies editing, the editing does not cause special treatment of the value; instead, the literal is treated as if the item had a simple alphanumeric description.
3. All numeric literals in a VALUE clause of an item must have a value which is within the range of values indicated by the PICTURE clause; for example, for PICTURE PPP99, the literal must be within the range .00000 to .00099.

4. The function of the BLANK WHEN ZERO clause or of any editing characters in a PICTURE clause has no effect on the initialization of the item. The VALUE clause is the only clause that may (depending on its usage) provide initialization. Editing characters are included, however, in determining the size of the item. Therefore, the value for an edited item must be presented in an edited form.

Format 2 of the VALUE clause is used only in condition-name entries. In a condition-name entry, the VALUE clause is required; the VALUE clause and the condition-name itself are the only two entries permitted. Both the conditional variable and the condition-name entries may have VALUE clauses.

Some of the possible ways of writing condition-name entries are:

```
nn data-name.
88 condition-name-1 VALUE IS literal-1.
88 condition-name-2 VALUES ARE literal-2, literal-3.
88 condition-name-3 VALUES ARE literal-4 AND
 literal-5 AND...
88 condition-name-4 VALUES ARE literal-6 THRU literal-7.
```

If the VALUE clause appears in an entry at the group level, the literal must be a figurative constant or a nonnumeric literal. The literal is aligned without consideration for the subordinate items belonging to the group. The VALUE clause cannot be stated at the subordinate levels within this group.

Under certain conditions, a data entry must not contain a VALUE clause:

1. The VALUE clause must not appear in a data description entry that contains an OCCURS clause or a REDEFINES clause.
2. Except in a condition-name entry, the VALUE clause must not appear in an entry that is subordinate to an entry that contains an OCCURS, REDEFINES, or VALUE clause.
3. The VALUE clause cannot be used for an item whose USAGE IS INDEX. Such an item cannot be a conditional variable.
4. The VALUE clause must not be specified for a group item containing items with different usages, or for a group containing items which are synchronized, justified, or which have usages other than DISPLAY.
5. Within a given record description, the VALUE clause must not be stated in a data description entry that is subsequent to a data description entry in which an OCCURS clause with a DEPENDING ON phrase appears.

#### GROUPS OF ELEMENTARY ITEMS

Data items that are further subdivided are called group items. A group consists of a sequence of subordinate groups and/or elementary items. Refer to the COBOL Reference Manual (Format 3 of the data description skeleton) for a description of the specific format entries.

The group record format is used for group item record description entries. The following rules must be observed:

1. Level-number must be an integer in the range 1-49 or 77.
2. If the REDEFINES clause is specified, it must be written immediately after data-name-1. The other clauses may be written in any order.
3. The OCCURS clause cannot be used with level-number 01.

A group entry may consist of only the level-number and data-name or FILLER, with all descriptive clauses omitted. The BLANK WHEN ZERO, JUSTIFIED, PICTURE, SYNCHRONIZED, and editing clauses are permitted in an elementary record description entry but must not be specified in a group record description entry.





## SECTION V

### FILE PROCESSING

#### FILE PROCESSING CONCEPTS

##### File Declaration

Every data file to be processed by a COBOL program requires source program statements in the Environment Division, Data Division, and Procedure Division.

A SELECT sentence for the file must be written in the FILE-CONTROL paragraph of the Environment Division.

Various options relating to the file's format and processing techniques may be specified in the I-O-CONTROL paragraph of the Environment Division.

An FD entry for the file must be written in the File Section of the Data Division, unless the file has been described implicitly via the RENAMING phrase in the SELECT sentence.

Record description entries must follow the FD entry, fully describing each data record mentioned in the FD entry. The records may be described in any order. Each data record's description must begin with a level 01 entry.

If USE procedures are employed in connection with label manipulation, peripheral device errors, or report group presentations, each USE procedure must appear as a separate section in the declarative portion at the beginning of the Procedure Division.

##### Sequential-Access Processing

The sequential-access technique applies to files stored on all types of media such as magnetic tape, cards, and mass storage devices. Except for mass storage devices, the sequential accessing of records has been the usual technique available for file processing. This technique is also applied to mass storage devices such as disk storage subsystems. Since mass storage devices also have random-access capabilities, their use for sequential access must be designated by phrases in the FILE-CONTROL paragraph of the Environment Division. To establish that a mass storage file is to be accessed sequentially (similar to tape file processing), the ACCESS MODE IS SEQUENTIAL and PROCESSING MODE IS SEQUENTIAL phrases are required in the SELECT sentence for the file.

With the sequential-access technique, the READ statement accesses the next logical record from the file and the WRITE statement releases a logical record for output. Therefore, the logical records appear on the peripheral device in the order in which they are read, processed, and/or written.

### Random-Access Processing

The random-access technique applies only to files that have been assigned to mass storage devices such as disk storage subsystems. The ACCESS MODE IS RANDOM and PROCESSING MODE IS SEQUENTIAL phrases in the FILE-CONTROL paragraph establish that a random-access technique is to be applied to the manipulation of records on that file.

The random-access technique differs from the sequential-access technique in that references to logical records in the file are not necessarily in the order of their appearance on the external device. Rather than spending time passing over records of a file which may not be applicable to the current problem, the user directly accesses a logical record in the file by specifying a key value associated with that logical record. The ACTUAL KEY phrase in the FILE-CONTROL paragraph provides the data-name of the field containing the key value; the phrase must be specified for random-access files.

Series 60/6000 COBOL requires that the data-name specified in the ACTUAL KEY phrase must be defined in a level 01 or level 77 entry in the Working-Storage Section as a single-precision binary integer. That is, the data-name must be described with USAGE COMPUTATIONAL-3 or with USAGE COMPUTATIONAL-1 with a PICTURE containing eight or less digits. The value of the actual key data item must indicate the relative position of the logical record within the file. The positioning of logical records is relative to the value zero, although the use of the value zero itself is not recommended. The user is responsible for controlling the contents of the ACTUAL KEY field.

The READ and WRITE statements access a specific addressable record from a mass storage file. In Series 60/6000 COBOL, there is one addressable record in a block. The record size may be adjusted within limits for a mass storage device by using the BLOCK CONTAINS clause in the file description entry for the file.

### Open Status and Closed Status

A file is either open or closed at any given time. A file is open if an OPEN statement has been executed and no CLOSE statement without the REEL option has intervened; otherwise, it is closed. Therefore, a file is implicitly in the closed status at the start of object program execution, and returns to the closed status when it is explicitly closed.

The following rules associate the file processing statements to the open or closed status of a file:

1. A file mentioned in an OPEN statement must be in the closed state when the OPEN statement is executed.
2. A file mentioned in any other input-output statement (such as READ, WRITE, or CLOSE) must be in the open state when the statement is executed.
3. The contents of a file may not be referenced except when the file is open.

4. Any closed file can be opened, unless it has been closed with lock. This means that a file can be reopened after it has previously been processed and closed.
5. Only one file of a multiple file tape can be in the open state at any given time.

### Input, Output, and I-O Modes

The OPEN statement specifies whether each file is to be opened in the INPUT mode, the OUTPUT mode, or the I-O mode. The mode of a file that is not open is indeterminate. Only mass storage files may be opened in the I-O mode.

The following rules associate the file processing statements to the input, output, or input-output status of a file:

1. A READ statement may refer only to an open input file or to an open I-O file.
2. A WRITE statement may refer only to records of an open output file or to records of an open I-O file.
3. If a file is closed, it may be opened either as input, output, or input-output. Thus, a file can be processed as output, closed, and then reopened for processing as input. (If desired, the file can then be closed, reopened as output, and the old data values can be overwritten with new values.)
4. The I-O mode cannot be used to initially establish a file, but rather may only be used to manipulate a file that has previously been established in the OUTPUT mode.
5. On a file that is open in the I-O mode, the records may only be modified; no insertions or deletions are allowed. To modify a record, procedures are inserted between the READ and WRITE statements. A READ statement followed immediately by another READ statement does not affect the contents of the first record on the mass storage file.

### Special File Processing

#### PROCESSING OPTIONAL FILES

A file may be optional in the sense that it may or may not be present for processing, depending upon an option specified at object program execution. COBOL has a special provision for processing optional input files but not for optional output files.

An optional input file must be described as such using the OPTIONAL phrase in the FILE-CONTROL paragraph. At program execution, the OPEN statement for the file verifies whether a file control card has been included for the file, among the other GCOS control cards. If the card has been included, the file is understood to be present, and the file is processed in the normal manner. If the card is omitted, the file is understood to be omitted, and the OPEN statement activates the end-of-file condition instead of attempting the normal open actions; the first READ statement produces no data record, but instead causes the relevant AT END or INVALID KEY procedure to be executed; and a subsequent CLOSE statement restores the file to the closed status.

To establish a conditionally existent output file, the user must provide explicit tests to determine whether or not the file is to be produced at program execution. The simplest way is to associate the file with one of the software switches provided by the operating system (see Section VI). The switch-status condition test may be used. Just before execution reaches the relevant OPEN statement, test the switch. If it is ON, continue processing, producing the file as usual. If the switch is OFF, execute ALTER statements, causing appropriate GO TO statements to bypass the OPEN, MOVE, WRITE, and CLOSE statements referring to the file.

#### PROCESSING NONLABELED MULTIPLE REEL FILES

COBOL rules state that a file must not be read after an AT END return unless it has first been closed and then reopened. The compiler itself cannot detect violations of this nature, so no such check is attempted. A special feature makes the use of READ statements after AT END phrases very important under certain circumstances.

Consider a multiple reel file on which label records are omitted. When the object program is executed, the final reel cannot be identified automatically from the tape contents. However, this condition can be determined with COBOL procedures. First, ascertain how many reels are expected; a suitable method would be to obtain a parameter giving the reel count from an ACCEPT statement. The following example illustrates how to achieve proper tape swapping and true end-of-file detection.

Example:

```
.
. A. READ file-name AT END GO TO EOF-CHECK.
. EOF-CHECK. SUBTRACT 1 FROM REEL-COUNT; IF REEL-COUNT IS NOT
ZERO GO TO A ELSE CLOSE file-name
GO TO ENDING-ROUTINE.
```

When the physical end-of-file mark at the end of each reel of the file is reached, it will result in the execution of the AT END procedure for the active READ statement. Execution of another READ statement then causes a tape swap, and obtains the first data record from the next reel. Execution of the CLOSE statement, on the other hand, entails standard nonlabeled file closeout conventions.

A nonlabeled tape is often terminated simply with a physical end-of-file mark. For a multiple reel file which is so constructed, the technique outlined above is the only practical way to accomplish tape swapping. If, however, each reel is terminated with a special data record whose value signals end of reel, tape swapping can be accomplished via the CLOSE REEL feature, used in connection with suitable IF tests.

Tape swapping via READ statements executed after AT END is a special feature provided by the Series 60/6000 software, and may not be available on other computers.

#### PROCESSING STRANGER FILES VIA COBOL

A stranger file is one which cannot be fully described to COBOL; the possible reasons are:

1. The character set of the data as stored on the peripheral device does not coincide with the character sets available to Series 60/6000 COBOL.
2. Data records are not arranged to occupy multiples of six characters.
3. Binary fixed- or floating-point numbers were produced on a computer system using signed magnitude arithmetic or exponent sizes that differ from Series 60/6000 requirements.
4. Variable-length records do not have record control words in the format required by the Series 60/6000 software; that is, variable-length records appear on a BCD mode tape.
5. Bit-coded parameters exist in the file.

A file with any of the properties listed above is probably produced by or for a computer other than the Series 60/6000 computers. Files with any of these properties, or many others, are nevertheless accessible to COBOL object programs.

GFRC provides dual entry point names (SYMDEFs) for many important functions:

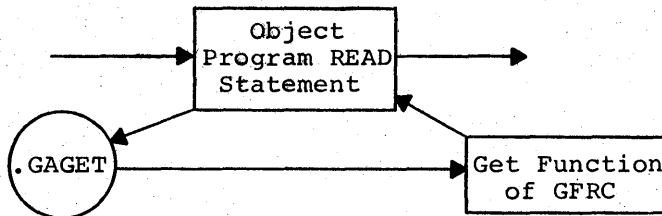
| <u>Standard Symbol</u> | <u>Alternate Symbol</u> |
|------------------------|-------------------------|
| OPEN                   | .GAOPE                  |
| CLOSE                  | .GACLS                  |
| GET                    | .GAGET                  |
| PUT                    | .GAPUT                  |
| PUTSZ                  | .GAPTS                  |
| WTREC                  | .GAWTR                  |
| IOEDIT                 | .GAEDI                  |
| PRINT                  | .GAPRN                  |
| PUNCH                  | .GAPNC                  |

The two symbols for each entry point are entirely equivalent. By convention, however, COBOL object programs utilize the alternate symbols shown above rather than the standard symbols.

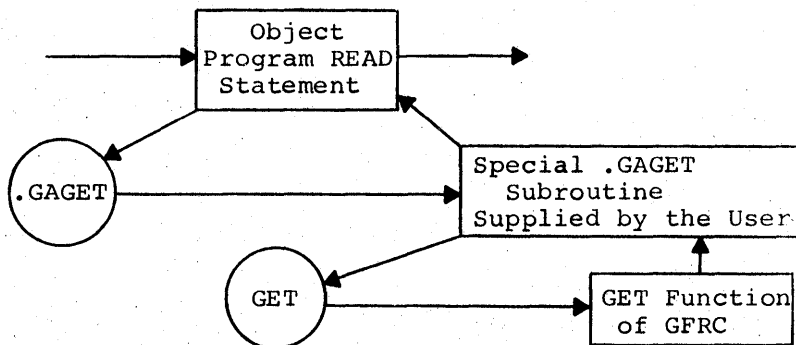
Each alternate symbol utilized by a COBOL object program defines a point at which a special subroutine may be inserted between the COBOL object program and the GFRC module. The insertion is accomplished by using the appropriate GFRC alternate symbol as the entry symbol for the special subroutine. When such a subroutine is present, it automatically intercepts all object program calls to the GFRC entry point in question.

For example, consider the GET function of GFRC, used for COBOL READ statements. The object time execution sequence may be diagrammed for various circumstances.

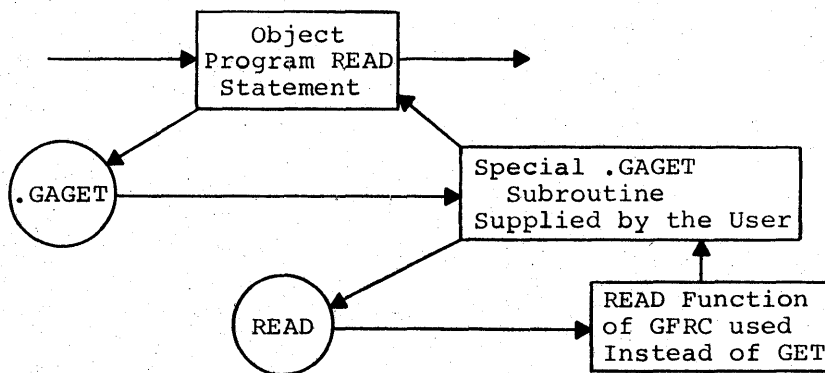
1. Normally, no special subroutine is present.



2. A simple format conversion subroutine supplied by the user might require GET.



3. A complicated stranger file format might require the special .GAGET subroutine to use GFRC services other than GET; when the object program calls the subroutine .GAGET, it does not engage GET.



A special subroutine of this kind is developed via GMAP as a relocatable binary subroutine, having as its entry symbol the appropriate GFRC alternate entry symbol.

#### Relationship of Reporting Verbs to File Processing

Reporting statements are closely related to WRITE statements. Each report described in the Report Section of the Data Division must be referenced in an FD statement in the File Section. This indicates that the report is to be produced on the referencing file. Execution of INITIATE, GENERATE, and TERMINATE statements referring to the report then causes report lines to be written to the file, just as WRITE statements cause data records to be written to a file. Therefore, a file which is to receive a report must be open as output whenever a relevant reporting statement is executed.



Summary of File Property Relationships

The following charts summarize the relationships of the various options for COBOL files.

| Property                                                | Implies or Requires |                            |                               |             |              |                  |                        |                        |                  |                     |              |
|---------------------------------------------------------|---------------------|----------------------------|-------------------------------|-------------|--------------|------------------|------------------------|------------------------|------------------|---------------------|--------------|
|                                                         | BLOCK SERIAL        | FLR (fixed-length records) | VLR (variable-length records) | BINARY mode | HIGH DENSITY | LABEL...STANDARD | BLOCK size ≤ 320 words | System Standard Format | Card Medium Code | Printer Medium Code | Process Area |
| SEQUENTIAL ACCESS                                       |                     |                            | I                             | R           |              |                  |                        | I                      |                  |                     | I            |
| RANDOM ACCESS                                           |                     | I                          |                               | R           |              |                  |                        |                        |                  |                     | I            |
| FOR CARDS                                               |                     |                            | I                             | R           |              |                  |                        | I                      | I                |                     | I            |
| FOR LISTING                                             |                     |                            | I                             | R           |              |                  |                        | I                      |                  | I                   | I            |
| BLOCK SERIAL                                            | I                   |                            |                               | R           |              |                  |                        |                        |                  |                     |              |
| SYSTEM STANDARD                                         | I                   |                            | I                             | R           | R            | R                | R                      | I                      |                  |                     |              |
| VLR (variable length records)                           |                     |                            | I                             | R           |              |                  |                        |                        |                  |                     |              |
| RECORDING MODE omitted                                  |                     |                            |                               | I           | I            |                  |                        |                        |                  |                     |              |
| BINARY HIGH DENSITY                                     |                     |                            |                               | I           | I            |                  |                        |                        |                  |                     |              |
| BCD (either density)                                    |                     | R                          |                               |             |              |                  |                        |                        |                  |                     |              |
| BLOCK clause omitted                                    |                     |                            |                               |             |              |                  | I                      |                        |                  |                     |              |
| VALUE OF label item                                     |                     |                            |                               |             |              | R                |                        |                        |                  |                     |              |
| Single DATA RECORD                                      |                     | I <sup>1</sup>             |                               |             |              |                  |                        |                        |                  |                     |              |
| Several DATA RECORDS, sizes unequal                     |                     |                            | I                             | R           |              |                  |                        |                        |                  |                     |              |
| REPORT[S]                                               |                     |                            | I                             | R           |              |                  |                        |                        |                  | I                   |              |
| REPORT[S] and DATA RECORD[S]                            |                     |                            | I                             | R           |              |                  |                        |                        |                  | I                   | I            |
| USAGE not DISPLAY[-1]                                   |                     |                            |                               | R           |              |                  |                        |                        |                  |                     |              |
| OCCURS...DEPENDING                                      |                     |                            | I                             | R           |              |                  |                        |                        |                  |                     | I            |
| USAGE COMPUTATIONAL[-n] requiring double-word precision |                     |                            |                               |             |              |                  |                        |                        |                  |                     | I            |
| SAME RECORD AREA                                        |                     |                            |                               |             |              |                  |                        |                        |                  |                     | I            |

<sup>1</sup> FLR is implied in the absence of overriding properties.

| Specified Property            | Conflicts With    |                         |             |              |                 |                               |               |                        |                     |                    |                      |                 |                    |                  |                 |                     |                              |           |                              |                          |                    |                     |
|-------------------------------|-------------------|-------------------------|-------------|--------------|-----------------|-------------------------------|---------------|------------------------|---------------------|--------------------|----------------------|-----------------|--------------------|------------------|-----------------|---------------------|------------------------------|-----------|------------------------------|--------------------------|--------------------|---------------------|
|                               | SEQUENTIAL ACCESS | RANDOM ACCESS FOR CARDS | FOR LISTING | BLOCK SERIAL | SYSTEM STANDARD | VLR (variable-length records) | MULTIPLE FILE | RECORDING MODE omitted | BINARY HIGH DENSITY | BINARY LOW DENSITY | BCD (either density) | BLOCK...RECORDS | BLOCK...CHARACTERS | LABEL...STANDARD | LABEL...OMITTED | VALUE OF label item | Several DATA RECORD, sizes # | REPORT[S] | REPORT[S] and DATA RECORD[S] | USAGE COMPUTATIONAL [-n] | OCCURS...DEPENDING | PARTITIONED RECORDS |
| SEQUENTIAL ACCESS             |                   | X                       | X           | X            |                 |                               | X             |                        | X                   | X                  |                      |                 |                    |                  | X               |                     |                              | X         | X                            |                          |                    |                     |
| RANDOM ACCESS                 | X                 |                         | X           | X            | X               | X                             | X             |                        | X                   | X                  |                      |                 |                    |                  |                 |                     |                              | X         | X                            |                          |                    | X                   |
| FOR CARDS                     | X                 | X                       | X           |              |                 |                               |               |                        |                     | X                  |                      |                 |                    |                  |                 |                     |                              | X         | X                            | X                        | X                  | X                   |
| FOR LISTING                   | X                 | X                       | X           |              |                 |                               |               |                        |                     | X                  |                      |                 |                    |                  |                 |                     |                              |           |                              | X                        | X                  | X                   |
| BLOCK SERIAL                  |                   |                         |             |              |                 |                               |               |                        |                     | X                  |                      |                 |                    |                  |                 |                     |                              |           |                              |                          |                    | 2                   |
| SYSTEM STANDARD               |                   | X                       |             |              |                 |                               |               |                        | X                   | X                  |                      |                 |                    |                  | X               |                     |                              |           |                              |                          |                    |                     |
| VLR (variable-length records) |                   | X                       |             |              |                 |                               |               |                        |                     | X                  |                      |                 |                    |                  |                 |                     |                              |           |                              |                          |                    | 3                   |
| MULTIPLE FILE                 | X                 | X                       |             |              |                 |                               |               |                        |                     |                    |                      |                 |                    |                  |                 |                     |                              |           |                              |                          |                    |                     |
| BINARY LOW DENSITY            | X                 | X                       |             | X            |                 |                               | X             | X                      | X                   |                    |                      |                 |                    |                  |                 |                     |                              |           |                              |                          |                    |                     |
| BCD (either density)          | X                 | X                       | X           | X            | X               | X                             | X             | X                      | X                   |                    |                      |                 |                    |                  |                 |                     | X                            | X         | X                            | X                        | X                  | X                   |
| BLOCK...RECORDS               |                   |                         |             |              |                 |                               |               |                        |                     | X                  | X                    |                 |                    |                  |                 |                     | 1                            | 1         | 1                            |                          | 1                  |                     |
| LABEL...OMITTED               | X                 |                         |             | X            |                 |                               |               |                        |                     |                    |                      |                 |                    | X                | X               |                     |                              |           |                              |                          |                    | X                   |
| VALUE OF label item           |                   |                         |             |              |                 |                               |               |                        |                     |                    |                      |                 |                    |                  | X               |                     |                              |           |                              |                          |                    |                     |
| Several DATA RECORD, sizes #  |                   |                         |             |              |                 |                               |               |                        |                     | X                  | 1                    |                 |                    |                  |                 | X                   |                              |           |                              |                          |                    |                     |
| REPORT[S]                     | X                 | X                       | X           |              |                 |                               |               |                        |                     | X                  | 1                    |                 |                    |                  |                 |                     |                              |           |                              | X                        |                    |                     |
| REPORT[S] and DATA RECORD[S]  | X                 | X                       | X           |              |                 |                               |               |                        |                     | X                  | 1                    |                 |                    |                  |                 |                     |                              |           |                              | X                        |                    |                     |
| USAGE COMPUTATIONAL [-n]      |                   |                         | X           | X            |                 |                               |               |                        |                     | X                  |                      |                 |                    |                  |                 |                     |                              | X         | X                            |                          |                    |                     |
| OCCURS...DEPENDING            |                   |                         | X           | X            |                 |                               |               |                        |                     | X                  | 1                    |                 |                    |                  |                 |                     |                              |           |                              |                          |                    | X                   |

- 1 Use of the CHARACTERS option is preferred when record sizes are not uniform.
- 2 Must also apply VLR and binary mode.
- 3 Must also apply Block Serial and binary mode.

## ASSIGNMENT OF FILES

### File Control Cards

Each COBOL file-name must be explicitly assigned to one or more two-character symbolic file-codes. When the object program is scheduled for execution, each file-code must be associated with a peripheral device via a file control card. There will be one such card for each file selected.

When used with COBOL object programs, each file control card must contain the file-code to which the file is assigned, explicitly or implicitly. Most cards other than \$ DATA, \$ PRINT, \$ PUNCH, \$ READ, and \$ SYSOUT require more parameters to provide additional information about the peripheral device and the mode of processing.

Files intended only for \$ DATA, \$ SYSOUT, or linked mass storage must have system standard format. Other files entering or leaving the system via bulk media conversion normally utilize magnetic tape as the intermediate device, and consequently require \$ TAPE cards. Any file having a block size exceeding 320 words must utilize magnetic tape or a random-access mass storage device.

Each file must be processed in a manner consistent with the nature of the peripheral device. Files intended for \$ PRINT, \$ PUNCH, or \$ SYSOUT must be output files and those intended for \$ DATA or \$ READ must be input files. If \$ TAPE is used, the file may be processed as either input or output and, after it is closed, it may be reopened as either input or output regardless of the method in which it was previously processed. This capability is particularly useful when a scratch file is needed within a program; the file may be processed as output and then be reopened and processed as input within the same program.

A mass storage file may be either input, output, or input-output. When the file status is input-output, the preferred file control card is \$ FILE or \$ PRMFL. The permanent data files must have both read and write permissions. If \$ DATA or \$ TAPE cards are used for the input-output mode, an abort will occur.

To engage the Incremental Report Printing (backdoor file) facility, specify the \$ USE, \$ FILE, \$ FUTIL, and \$ DATA cards as described in Section XVI.

For additional information concerning file control cards and deck setups, refer to the Control Cards reference manual and to Appendix B.

### System Standard Format

When the APPLY SYSTEM STANDARD FORMAT phrase in the I-O-CONTROL paragraph has been specified for a file, a substantial degree of peripheral device independence is obtained at the expense of processing efficiency. The file may be assigned to different peripheral devices from one execution of the object program to the next. The following example illustrates the potential flexibility of a daily report file:

1. On Monday, the file is assigned to an online printer.
2. On Tuesday, the file is assigned to mass storage, and a subsequent media conversion to printer is scheduled.

3. On Wednesday, the file is assigned to magnetic tape, and again a subsequent media conversion is scheduled.
4. On Thursday, the file is assigned to SYSOUT, so that subsequent media conversion is automatically scheduled.

If a file is ultimately intended for a printer or a card device, the file may be visualized in terms of the selected device, even though an intermediate conversion to or from tape or mass storage may be desired. In this case, the FOR CARDS or FOR LISTING option should be specified in the SELECT sentence of the FILE-CONTROL paragraph. System standard format will then be automatically applied and the records will be identified as print-line or card images with the appropriate media codes.

A Series 60/6000 data file having system standard format has the following properties:

1. Data blocks may vary in length, not exceeding 320 words, including block serial numbers.
2. Block serial numbers are applied.
3. The recording mode is binary high density.
4. Variable-length record (VLR) format is applied.
5. Label records are standard.

Files having logical records larger than the block size of 320 words may nevertheless be processed in system standard format by the operating system if the requirements for partitioned records are met. (Refer to the Partitioned Records paragraph in Section II.)

When system standard format is used, the BLOCK CONTAINS and RECORDING MODE clauses may be omitted, and it is unnecessary to specify APPLY VLR or APPLY BLOCK SERIAL NUMBER. However, according to COBOL rules, the LABEL RECORDS clause must appear in every FD entry in the File Section.

To increase efficiency, files may deviate from the system standard format. For example, a large master file which is assigned to magnetic tape might use block sizes larger than 320 words. A block size three times as large (960 words) would increase the capacity of a tape by approximately 20 percent.

The system standard format must be used when device independence is intended.

Label records, although conceptually standard, do not actually appear in mass storage.

System standard format must be used for files intended for system output (SYSOUT).

## Peripheral Devices

Series 60/6000 COBOL files may be assigned to disk, magnetic tape, card reader, card punch, or printer peripheral devices.

The preferred media for object program access are the disk and magnetic tape devices, since the operating speed of this equipment is better matched to the capability of the central processor. Although the printer, card reader, and card punch can be accessed directly by object programs, it is preferable to utilize the media conversion procedure, so that a file intended for cards or listing nevertheless uses a high-speed peripheral device when the object program is executed. Standard media conversion programs are provided as part of the software system.

In Series 60/6000 COBOL, the SELECT sentence in the FILE-CONTROL paragraph is used to assign each file to a symbolic file-code, not to a specific peripheral device. In this case, the actual device is indicated separately with control cards when the object program is scheduled for execution.

If a file is intended for a printer or a card device, the FOR LISTING or FOR CARDS option should be specified in the SELECT sentence of the FILE-CONTROL paragraph. Specification of the FOR LISTING or FOR CARDS option automatically implies system standard format.

Some data description options, such as USAGE DISPLAY-2 or USAGE COMPUTATIONAL (or COMPUTATIONAL-n), are not suitable for card or printer files.

It is preferable to use standard label records for all magnetic tape files, including those intended for cards or listing; however, the user may wish to describe and process label records which are more meaningful to the specific computer installation.

## Multiple File Tapes

The MULTIPLE FILE phrase in the I-O-CONTROL paragraph applies only to magnetic tapes. It permits two or more files to appear successively on the same physical reel of tape. For COBOL object programs, all files of a multiple file tape must actually be present. The tape is automatically positioned to the proper point each time a file is opened (as either input or output). Multiple file tape positioning is based on counting the number of logical files intervening between the desired file and the beginning of the tape, not on a label search. Therefore, a file in position 5 must be preceded on the tape by four prior files.

On an output tape, files must actually be written in the order in which they are to appear on the tape. For example, the position 4 file cannot be written before the position 2 file. However, the successive files may be produced by separate object programs. On an input tape, files may be processed in any order.

Two files on a multiple file tape cannot be open concurrently. When an output file in a given position has been opened, any data which might have been previously recorded beyond that point on the tape is unavailable.

All of the files on a multiple file tape must have the same recording mode.

Either all of the files on a multiple file tape must be labeled, or else none may be labeled.

Since each file referenced in the MULTIPLE FILE TAPE phrase must be assigned to a unique file-code in the ASSIGN phrase of the SELECT sentence, the different file-codes thus assigned for files contained on the same multiple file tape must then be assigned to the same logical unit designator (LUD) by \$ TAPE control cards when the object program is to be executed.

Example:

```
INPUT-OUTPUT SECTION.
FILE-CONTROL.
 SELECT FILE-C ASSIGN TO T1.
 SELECT FILE-A ASSIGN TO T2.
 SELECT FILE-B ASSIGN TO T3.
I-O-CONTROL.
 SAME AREA FOR FILE-B FILE-C FILE-A.
 MULTIPLE FILE TAPE CONTAINS FILE-A FILE-B FILE-C.
```

Control Cards Example:

```
$ TAPE T1,A2S
$ TAPE T2,A2S
$ TAPE T3,A2S
```

When the files are labeled, the tape layout is as follows:

```
Beginning of tape indicator (load point reflective foil)
BEGINNING-FILE-LABEL
End-of-file mark
Data blocks
End-of-file mark
ENDING-FILE-LABEL
End-of-file mark
BEGINNING-FILE-LABEL
End-of-file mark
Data blocks
End-of-file mark
ENDING-FILE-LABEL
End-of-file mark
Etc.
```

} First file

} Second file

When the files are not labeled, the tape layout is as follows:

```
Beginning of tape indicator
Data blocks
End-of-file mark
Data blocks
End-of-file mark
Etc.
```

} First file

} Second file

COBOL rules imply that all of the files on a multiple file tape must actually fit on one physical reel of tape.

## FILE PROCESSING AREAS

### Buffer Areas

At least one buffer area must be available for each file in the object program. Two buffer areas are automatically assigned to each file selected in the source program.

A single buffer area may be allocated to any file by using the RESERVE NO ALTERNATE AREA phrase in the SELECT sentence. If only one buffer is employed, input or output operations for the file cannot proceed concurrently with the procedural manipulation of its records in the object program. During input-output operations on such a file, control is therefore passed to the operating system to allow other active object programs to utilize the central processor. However, if an alternate buffer is reserved, the object program can continue to process data in the current buffer concurrently with the execution of input-output operations involving the alternate buffer. Alternating buffers can therefore contribute to object program efficiency, particularly when high-volume files are being processed. The decision to buffer a file should be based upon considerations such as the size and activity of the file and the effective use of memory within a multiprogramming environment.

A buffer area, or a pair of alternating buffer areas, can be shared by two or more files by specifying the SAME AREA phrase in the I-O-CONTROL paragraph. The compiler then evaluates the maximum buffer requirements of all of the files, and allocates adequate aggregate buffer space to the first file to handle any of the other files, but the single buffering or double buffering for each file is still determined by the RESERVE phrase of each individual file.

The SAME AREA phrase should not be used for files which are open concurrently.

### Record Areas

The APPLY PROCESS AREA phrase in the I-O-CONTROL paragraph is used as an alternate method of processing files. A process area is a memory area outside the buffers, and of sufficient size to hold the largest logical record of a file. When a file using a process area is read, each logical record is implicitly moved to the process area from the input buffer when the READ statement is executed, and all subsequent procedures refer to the record in the process area. When such a file is written, each logical record is implicitly moved from the process area to the output buffer when the WRITE statement is executed, and all subsequent procedures similarly refer to the record in the process area.

A process area may be applied regardless of the number of buffer areas allocated. Several source language options direct the compiler to apply a process area:

1. APPLY PROCESS AREA
2. FOR CARDS or FOR LISTING
3. Random access or sequential access
4. OCCURS...DEPENDING

5. SAME RECORD AREA (see below)
6. Both REPORTS and DATA RECORDS (the data records utilize the process area)
7. Double-precision COMPUTATIONAL(-n) data items in records
8. Sort or merge files

When no process area is applied, explicitly or implicitly, the current data record is processed in the buffer. Since the origination of records in the buffer usually differs from one record to another, the record contents must be addressed relatively with respect to the current record origin. Extra housekeeping is required when processing such a record, because of the necessity for relative addressing.

A considerable savings in both space and time may often be realized by applying a process area on those files whose data records are involved in a large number of procedural references in a program. These economies are attributable to the simplification of the object code generated for each of the references to a fixed process area without the indirect addressing and related overhead involved in referencing the record in the buffer area. Usually these economies in data reference will overshadow the amount of time required to move the records between the buffer and process area. However, the WRITE...FROM and READ...INTO statements are not efficient when used with a process area since a double move of the data would result. That is, these statements would cause a move from the buffer to the process area followed by a move from the process area to the identifier referenced in the READ...INTO statement in the case of an input file, or a move from the identifier referenced in the WRITE...FROM statement to the process area followed by a move from the process area to the buffer in the case of an output file.

A process area must be applied, explicitly or implicitly, for files with logical records larger than 320 words and which require the partitioned record capability of the operating system.

When a process area is used, more effective blocking on variable-length record output files occurs. If no process area is applied to such files, each output block is physically written when the remaining buffer space is not of sufficient size to hold the largest record of the file; thus, actual block sizes can be substantially smaller than the maximum block size of the file. When a process area is applied, however, the output block is physically written only when the remaining area is too small for the current logical record. If a process area is used, therefore, one or more extra records can often be placed into an output block.

A process area may be shared by two or more files by specifying the SAME RECORD AREA phrase. The compiler then allocates a process area to the first file that is of sufficient size for the largest logical record of all of the files involved, and causes all of the files to share this process area. (In this case, the process area is applied whether or not it has been explicitly specified.) Files utilizing the FOR CARDS or FOR LISTING option may not be referenced in a SAME RECORD AREA phrase.

The SAME RECORD AREA phrase does not result in buffer sharing, but implies a shared process area. This phrase is recommended for minimizing moves on a master file that is being updated. If the SAME RECORD AREA phrase is specified for this purpose, special procedures must be used to avoid deleting input records when insertions are required.



A file may have one or more buffer areas in any object program, regardless of how many it may have in other object programs. Similarly, a file may participate in SAME AREA or SAME RECORD AREA phrases in a given program, whether or not it is involved in these options in other programs.

If the rules for modularization are followed, the compiler arranges interfaces so that a single set of memory areas is allocated to each file in a modularized program, and each subprogram refers to the same memory area at object program execution.

### Sort Areas and Sort-Merge Areas

The SAME SORT AREA and/or SAME SORT-MERGE AREA phrases allow the user to designate memory areas that may be shared, or reused, during the sorting or merging of several sort or merge files. In the implementation of the Series 60/6000 sort-merge feature, such optimization is automatically provided in that all the sort or merge processes in a run unit share a single common memory area.

In addition, the SAME SORT AREA and/or SAME SORT-MERGE AREA phrase allows the user to designate memory areas associated with non-sort (merge) files that may be used in the sorting or merging of specified sort or merge files to the extent defined by the implementor. In the Series 60/6000 sort-merge implementation, such sharing does not take place.

Since the functionality of the SAME SORT AREA and SAME SORT-MERGE AREA phrases is fully satisfied within the sort-merge implementation itself, the presence or absence of either of these phrases will have no functional effect on the program.

### FILE PROCESSING STATEMENTS

The following verbs (statements) describe file processing in the source program:

|                                        |   |                                                        |
|----------------------------------------|---|--------------------------------------------------------|
| OPEN<br>READ<br>WRITE<br>SEEK<br>CLOSE | } | File processing verbs                                  |
| USE                                    | } | Compiler-directing verb                                |
| ACCEPT<br>DISPLAY                      | } | Low-volume data transmission verbs<br>(see Section VI) |
| ACCEPT MESSAGE<br>DISPLAY              | } | Transaction Processing verbs<br>(see Section VII)      |
| INITIATE<br>GENERATE<br>TERMINATE      | } | Reporting verbs<br>(see Section VIII)                  |

SORT  
MERGE  
RELEASE  
RETURN

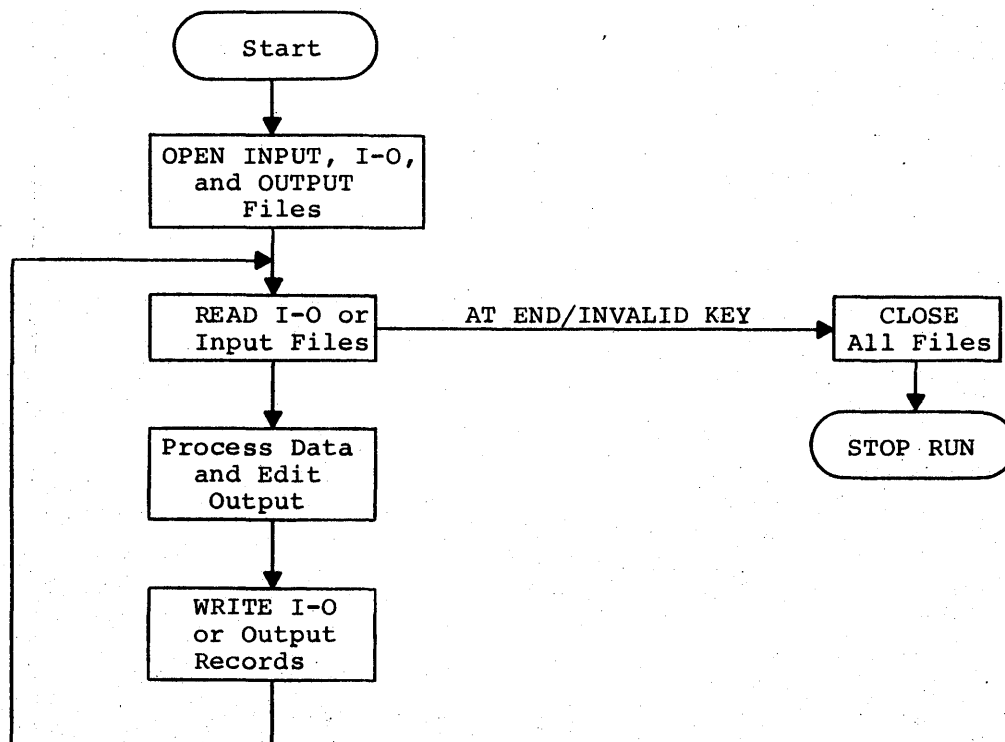
} File ordering verbs  
(see Section IX)

The functions of the file processing statements are briefly summarized below:

- OPEN Initiates the processing of each file, and provides initial rewinds, beginning label handling, and initialization of input buffer contents.
- READ Obtains a logical record from an input or I-O file, and executes a specified imperative-statement when an end-of-file or INVALID KEY condition is detected.
- WRITE Transmits a logical record to an output or I-O file for storage on a peripheral device.
- SEEK Initiates the accessing of a mass storage record, which is accomplished automatically in READ and WRITE statements.
- CLOSE Terminates the processing of each file, and provides ending label handling, as well as the final rewind and lock capability.

The USE statement specifies procedures for input-output label and error handling that serve as supplements to the standard procedures provided by the input-output system. It also provides the mechanism for specifying out-of-line procedural statements for processing mass storage files.

The relationship of the file processing statements is illustrated by a schematic flow chart of a COBOL program:



The file processing activities described in the preceding flow chart must conform to the following rules:

1. Each file must be explicitly opened with an OPEN statement before any other procedures (such as READ, WRITE, or CLOSE statements) referring to it are executed.
2. The first record of a file is available only after an initial READ statement has been executed.
3. Each READ statement must include an AT END or INVALID KEY provision, so that the input-output routine can signal to the object program when an invalid access occurs or the end of the file is reached.
4. Each file must be explicitly closed with a CLOSE statement after all processing on it is completed.
5. When the end-of-file condition is reached, the AT END operation is effected and no record is obtained.

### OPEN Statement

The OPEN statement in the Procedure Division is used to initiate the processing of files. An OPEN statement also causes label checking/writing and other input-output activities to be performed. Each of the format choices (INPUT, OUTPUT, I-O) may be specified only once in an OPEN statement.

#### Example:

```
OPEN INPUT FILE-D OUTPUT FILE-E I-O FILE-F.
```

The OPEN statement must not be used for sort files or merge files, but must be used for all other files. The OPEN statement for a file must be executed prior to the first READ, WRITE, or SEEK statement for that file.

A second OPEN statement for a file executed prior to a CLOSE statement for that file will cause a CK abort to occur.

The object program must not attempt to place data in the current record of an output file before the file has been opened. Failure to observe this rule may lead to unpredictable results in the NEIS mode and to an abort condition in the EIS mode. An OPEN statement for an output file, in addition to its other activities, establishes a memory area in which the first data record of the file can be built.

The OPEN statement does not obtain or release the first data record. A READ or WRITE statement must be executed to obtain or release, respectively, the first data record. Input data cannot be referenced until the first READ statement has been executed for the file. Unless an APPLY PROCESS AREA phrase has been specified for a file, no reference to the record area should be made until data is obtained.

For files described with the FOR LISTING option, the report is not automatically positioned at the top of the page when the OPEN statement is executed.

For a multiple file tape, an OPEN statement causes automatic positioning to the proper file. The positioning technique varies with different computers; for Series 60/6000, the positioning is based on counting files as they are bypassed.

If a label record is specified for a file, the label is processed according to the standard beginning label convention. If specified by the USE statement, the user's label procedure is executed. The order of execution of these two processes is specified by the USE statement. The behavior of the OPEN statement when a label record is specified but is not present, or when a label record is not specified but is present, is undefined.

When processing mass storage files for which the access mode is sequential, the OPEN statement supplies the initial value for the actual key associated, explicitly or implicitly, with the file.

#### INPUT OPTION

If an input file is designated with the OPTIONAL phrase, the object program causes an interrogation for the presence of this file. If the file is not present, the first READ statement for this file causes the imperative-statement in the AT END phrase to be executed.

#### I-O OPTION

The I-O phrase pertains only to mass storage files.

The I-O phrase permits the opening of a mass storage file for both input and output operations. Since this phrase implies the existence of the file, it cannot be used if the mass storage file is being initially created.

When the I-O phrase is specified and the LABEL RECORDS clause indicates that label records are present, the following procedures are included when the OPEN statement is executed:

- a. The label (if it exists) is checked in accordance with the standard conventions for input-output label checking.
- b. The user's beginning label procedure, if one is specified by the USE statement, is executed.
- c. The new label is written in accordance with the standard conventions for input-output label writing.

Only the current record of a file is available at any given time for either an INPUT, an OUTPUT, or an I-O file. Although several record types may have been defined for a file, only the information which is present in the current record is accessible to the program. As records are successively processed, they conceptually share the same memory area. If any executed procedural statement attempts to access information which is not part of the current record, the results are unpredictable. In addition, any attempt to access the current record when no current record exists, such as during or after execution of AT END or INVALID KEY procedures, may also yield unpredictable results.

## NO REWIND OPTION

The NO REWIND phrase does not apply to mass storage processing.

The NO REWIND phrase can be used only with a sequential single reel.

If the external device for the file permits rewinding, the following rules apply:

- a. When the NO REWIND phrase is not specified, execution of the OPEN statement causes the file to be positioned at its beginning.
- b. When the NO REWIND phrase is specified, execution of the OPEN statement does not cause the file to be repositioned. Therefore, when the NO REWIND phrase is specified, the file must have previously been positioned at its beginning.

## READ Statement

A file must be opened as either INPUT or I-O before a READ statement can be executed.

The READ statement processes only one logical record at a time, regardless of the method in which the file is blocked or buffered. An OPEN statement does not implicitly READ the first data record of an input file. Only one record of a file is available at a time; however, a record may be saved by moving it into working-storage.

For sequential-access file processing, the READ statement makes available the next logical record from an input or input-output file and allows a specified imperative-statement to be performed when the end-of-file condition is detected.

For the sequential-access technique, the READ statement requires the file-name as its operand rather than a record-name. The reason is that a file may have several record types, but READ delivers the next available record in sequence regardless of type. A READ statement that references a specific record-name is therefore illogical.

For the random-access technique, the READ statement delivers a logical record associated with the value in the ACTUAL KEY data-name for the file. If this value is found to be outside the number of links/llinks assigned to the file on the file control card, the INVALID KEY procedures are engaged by the input-output routine. Similarly, the WRITE statement causes a check to be made on the contents of the ACTUAL KEY before the logical record is written to determine if the INVALID KEY procedures are to be engaged. When an INVALID KEY condition exists, no writing takes place and the current record is available to the program.

Regardless of the method used to overlap access time with processing time, a record is available prior to the execution of any statement following the READ statement.

When a file has more than one record description, these records share the same memory area. Only the information that is present in the current record is accessible. An explicit test should follow each READ statement to determine the current record type. It is often convenient to plan the record formats of the file so that a single item, common to all records, can be tested to determine the current record type.

Example:

```
FD FILEA LABEL RECORDS ARE STANDARD.
01 REC-A.
 02 DN-1 PIC 9(4).
 02 DN-2 PIC X(120).
01 REC-B.
 02 DN-3 PIC 9(4).
 02 DN-4 PIC X(60).
 02 DN-5 PIC X(60).
```

The contents of DN-1 and DN-3 can be used to determine whether REC-A or REC-B has just been read.

Each READ statement must include an AT END or INVALID KEY phrase. The imperative-statement that follows each phrase may include one or more verbs and is delimited by a period.

INTO OPTION

The INTO phrase must not be used when the input file contains logical records of various sizes as indicated by their record descriptions. The memory area associated with 'identifier' and the record area associated with file-name must not be the same memory area. File-name must not represent a sort file or a merge file.

If the INTO phrase is specified, the current record is moved from the input area to the area specified by identifier according to the rules for the MOVE statement without the CORRESPONDING phrase. Any subscripting or indexing associated with identifier is evaluated after the record has been read and immediately before it is moved to the data item.

When the INTO phrase is used, the record being read is available in both the input record area and the data area associated with identifier.

AT END PHRASE

The AT END phrase is used for non-mass-storage files and for mass storage files in the sequential-access mode.

If, after reading the last logical record of a file, another READ statement is initiated for that file, that last logical record is no longer available in its record area and the READ statement is completed by the execution of the AT END phrase. After the AT END condition has been recognized for a file, a READ statement for that file must not be specified without prior execution of a CLOSE statement and an OPEN statement for that file. The logical end of the file is recognized when an end-of-file mark or physical end of unit has been encountered on the file.

The program must not execute statements referencing data items in an input file either before the initial READ statement for the file has been executed, or after the AT END phrase has been executed. Failure to observe this rule may lead to unpredictable results in the NEIS mode or to an abort condition in the EIS mode.

If a file described with the OPTIONAL phrase is not present, the imperative-statement in the AT END phrase is executed on the first READ. The standard end-of-file procedures are not performed.

If the end of a tape reel is recognized during execution of a READ statement, and the logical end of the file has not been reached, the following operations are performed:

- a. The system ending reel label procedure and the user's ending reel label procedure, if the latter procedure is specified by the USE statement. The order of execution of these two procedures is determined by the USE statement.
- b. A reel swap.
- c. The system beginning reel label procedure and the user's beginning reel label procedure, if the latter procedure is specified. The order of execution is again determined by the USE statement.
- d. The first data record of the new reel is made available.

#### INVALID KEY PHRASE

The INVALID KEY phrase is used for mass storage files that are processed in the random-access mode.

The READ statement implicitly performs the function of the SEEK statement for a specific mass storage file. If such a file is accessed for a specified mass storage record and the contents of the associated ACTUAL KEY data item are invalid, the INVALID KEY phrase is executed. The maximum value for an ACTUAL KEY data item is 262,142.

#### WRITE Statement

An OPEN statement specifying either OUTPUT or I-O must be executed for a file prior to referencing a data item in the file or executing the first WRITE statement for that file. The OPEN statement does not implicitly WRITE the first data record of an output file.

The WRITE statement is used to place a logical record on the file named in the associated file description. The record-name is the name of a logical record in the File Section of the Data Division and may be qualified. The record-name must not be part of a sort file or a merge file.

The WRITE statement requires a specific record-name, rather than a file-name, as its operand. The program steps leading to the WRITE statement may have prepared any record type defined for the file for output. Therefore, the exact record type that has been prepared must be specified in the WRITE statement to permit the appropriate housekeeping to occur in the output process.

The WRITE statement performs the following operations after recognizing the end-of-reel condition on files assigned to a magnetic tape:

- a. The system ending reel label procedure and the user's ending reel label procedure if the latter procedure is specified by the USE statement. The order of execution of these two procedures is specified by the USE statement.
- b. A tape swap. (This includes rewinding the completed reel and placing it in the standby condition.)
- c. The system beginning reel label procedure and the user's beginning reel label procedure if the latter procedure is specified by the USE statement. The order of execution of these two procedures is specified by the USE statement.

The previous data content of an output record is not available for further internal processing after the execution of a WRITE statement referencing that record unless a process area exists for the file. After a WRITE statement is executed, the current record of the file has yet to be built, and the data values within the current record are consequently unpredictable until the execution of additional statements causes new values to be transmitted to the current record. If a process area exists for the file, the WRITE statement may be used repetitively to duplicate records on the peripheral device without executing any intervening program statements.

#### FROM PHRASE

The FROM phrase causes the value of identifier to be implicitly moved to record-name (that is, to the current output record area). The identifier must be the name of a data item within the Working-Storage Section or in an input record area. If the format of the identifier differs from that of the record-name, moving will take place according to the rules specified for the MOVE statement without the CORRESPONDING phrase. Normally, the information in the record-name area is no longer available, but the information in the identifier area is available. It is illegal for record-name and identifier to be the same name.

#### ADVANCING PHRASE

The ADVANCING phrase provides control of the vertical positioning of each record (line) on the printed page. For printed output, vertical format control is provided for each line, either by the slewing of one line before printing, which is automatically provided for files described with the FOR LISTING option in the SELECT sentence, or by specifying the desired vertical slew control in the ADVANCING phrase. Otherwise, the output format will not meet printer requirements and the printout will be unsatisfactory. If the ADVANCING phrase is not specified, or the AFTER ADVANCING phrase is specified, the time required to print the report will be twice as long as that required if the BEFORE ADVANCING phrase is specified. In the ADVANCING phrase, the following rules apply:

- a. When a WRITE statement is executed, the value of identifier-2 will determine the number of lines the listing will be advanced. To avoid unnecessary numeric conversions, it is recommended that identifier-2 be described as a single-precision binary integer. (That is, it should be described with USAGE COMPUTATIONAL-3 or USAGE COMPUTATIONAL-1 with a PICTURE containing less than nine digits.)



- b. When integer is specified, it must be a nonnegative integer. The value of integer will determine the number of lines the listing will be advanced.
- c. TOP causes the listing to be advanced to the top of the page. If the mnemonic-name assigned to the special name 'TOP' is specified, the effect is the same as if the TOP OF PAGE option is used.
- d. The ADVANCING phrase may be used only when the FOR LISTING option has been specified in the SELECT sentence. It cannot be used for a file described with OCCURS...DEPENDING. The APPLY SYSTEM STANDARD FORMAT phrase should generally be specified for a file to which WRITE...ADVANCING is applied.
- e. When the ADVANCING mnemonic-name phrase is specified, any mnemonic-name defined in the SPECIAL-NAMES paragraph is acceptable.

The following example illustrates the line spacing of contiguous WRITE statements with and without the ADVANCING phrase:

| Present<br>WRITE                    | WRITE        | WRITE AFTER<br>ADVANCING<br>1 LINE | WRITE BEFORE<br>ADVANCING<br>1 LINE |
|-------------------------------------|--------------|------------------------------------|-------------------------------------|
| Previous<br>WRITE                   |              |                                    |                                     |
| WRITE                               | Single space | Single space                       | Overprint                           |
| WRITE AFTER<br>ADVANCING<br>1 LINE  | Single space | Single space                       | Overprint                           |
| WRITE BEFORE<br>ADVANCING<br>1 LINE | Double space | Double space                       | Single space                        |

The following example illustrates the page spacing of contiguous WRITE statements with the ADVANCING TO TOP OF PAGE phrase:

| Present<br>WRITE                            | WRITE AFTER<br>ADVANCING TO<br>TOP OF PAGE              | WRITE BEFORE<br>ADVANCING TO<br>TOP OF PAGE              |
|---------------------------------------------|---------------------------------------------------------|----------------------------------------------------------|
| Previous<br>WRITE                           |                                                         |                                                          |
| WRITE AFTER<br>ADVANCING TO<br>TOP OF PAGE  | One line printed<br>before advancing<br>to top of page. | Two lines printed<br>before advancing<br>to top of page. |
| WRITE BEFORE<br>ADVANCING TO<br>TOP OF PAGE | Blank page.                                             | One line printed<br>before advancing<br>to top of page.  |

## INVALID KEY PHRASE

The INVALID KEY phrase is used for processing mass storage files.

For mass storage files in the sequential-access mode, the imperative-statement in the INVALID KEY phrase is executed when the end of the allocated space is reached and an attempt is made to execute a WRITE statement for that file.

For files in the random-access mode, the imperative-statement in the INVALID KEY phrase is executed when the content of the actual key being used to obtain the mass storage record is found to be invalid. When an INVALID KEY condition exists, no writing takes place and the information in the record area is available. The maximum value for an ACTUAL KEY data item is 262,142.

## SEEK Statement

An explicit SEEK statement is not required for processing mass storage files. The function of the SEEK statement is performed implicitly by a READ or WRITE statement. The contents of the actual key are used to determine the relative location of the desired record within the file when the implicit seek function is executed.

## CLOSE Statement

All files that have been opened must be closed before a STOP RUN statement is executed.

A CLOSE statement is used to terminate the processing of reels and files, with optional rewind and/or lock capabilities available where applicable. A CLOSE statement must not reference a sort file or a merge file.

If the records of a file are blocked and/or buffered, some output data may remain in a memory buffer after the last WRITE statement has been executed for the file. The CLOSE statement causes such data to be written to the peripheral device, following the appropriate conventions in the case of partially filled data blocks. This condition emphasizes the requirement that each file be explicitly closed when processing is completed.

If a file has been specified with the OPTIONAL phrase in the FILE-CONTROL paragraph and this file is not present, the standard end-of-file processing is not performed.

For a multiple reel magnetic tape file, standard tape swap procedures are automatically applied to each reel except the last reel of the file, unless explicit CLOSE REEL statements intervene. The CLOSE statement for the overall file may affect the position of the final reel, but has no effect on the prior reels. Similarly, a CLOSE REEL statement affects only the current reel, not prior or subsequent reels of the file. A CLOSE REEL statement may refer only to a multiple reel magnetic tape file.

For files described with the FOR LISTING option, the report is not automatically positioned at the top of the page when the CLOSE statement is executed.

## STANDARD CLOSE FILE

If the file is allocated to magnetic tape, the current reel is rewound to its beginning.

If a CLOSE statement without the REEL option has been executed for a file, a READ, WRITE, or SEEK statement for that file must not be executed unless an intervening OPEN statement for that file is executed.

If the MULTIPLE REEL or MULTIPLE UNIT phrase was specified in the SELECT sentence of the FILE-CONTROL paragraph for a sequential-access mode input file, all reels in the file prior to the current reel are processed according to the standard reel swap procedure, except for those reels controlled by a prior CLOSE REEL statement. If the current reel is not the last in the file, the reels in the file following the current one are not processed in any manner.

If the MULTIPLE REEL or MULTIPLE UNIT phrase was specified in the SELECT sentence of the FILE-CONTROL paragraph for a sequential-access mode output file, all reels in the file prior to the current reel are processed according to the standard reel swap procedure, except for those reels controlled by a prior CLOSE REEL statement.

Standard Close File With Lock Option: An appropriate technique is supplied to ensure that this file cannot be opened again during this execution of this object program.

Standard Close File With No Rewind Option: The NO REWIND option applies only to files allocated to a magnetic tape. The current reel is left in its current position.

## STANDARD CLOSE REEL

The REEL option applies only to files stored on tape devices. The current reel is rewound to its beginning position.

Standard Close Reel With Lock Option: An appropriate technique is supplied to ensure that the current reel cannot be processed again as a part of this file during this execution of this object program. The current reel is rewound to its beginning position and the device is placed in the standby status.

Standard Close Reel With No Rewind Option: The current reel is left in its current position.

## USE Statement

The USE statement specifies procedures for input-output label and error handling that serve as supplements to the standard procedures provided by the input-output system. It is also used to specify Procedure Division statements that are executed just before a report group named in the Report Section of the Data Division is produced.

The USE statement in the Procedure Division provides the mechanism for specifying out-of-line procedural statements for processing mass storage files.

A USE statement, when present, must immediately follow a section header in the declarative portion of the Procedure Division and must be followed by a period followed by a space. The remainder of the section must consist of one or more procedural paragraphs that define the procedures to be used. The USE statement itself is never executed; rather, it defines the conditions calling for the execution of the USE procedures.

The same file-name can appear in a different specific arrangement of a format. However, the appearance of a file-name in a USE statement must not cause the simultaneous request for execution of more than one USE declarative.

A file-name representing a sort file or a merge file may not appear in a USE statement.

Within a USE procedure, there must be no reference to nondeclarative procedures. Conversely, in the nondeclarative portion, there must be no reference to procedure-names that appear in the declarative portion, except that PERFORM statements may refer to a USE declarative or to the procedures associated with such a USE declarative.

#### ERROR PROCEDURE PHRASE

The designated error procedures are executed after the standard input-output error procedures have been executed.

#### LABEL PROCEDURE PHRASE

The designated label procedures are executed:

- a. Before or after a beginning or ending input label check procedure is executed.
- b. Before a beginning or ending output label is created.
- c. After a beginning or ending output label is created, but before it is written.
- d. Before or after a beginning or ending input-output label check procedure is executed.

If the file-name phrase is used, the file description entry for each file-name must not specify a LABEL RECORDS ARE OMITTED clause.

If the words BEGINNING or ENDING are not included, the designated procedures are executed for both beginning and ending labels.

If the REEL or FILE option is not included, the designated procedures are executed for both REEL and FILE labels. The REEL phrase is not applicable to mass storage files.

Within the procedures of a USE declarative in which the USE statement specifies a phrase other than the file-name phrase, references to common label items need not be qualified by a file-name. A common label item is an elementary data item that appears in every label record of the program, but at the same time does not appear in any data record of this program. Furthermore, a common label item must have the same name, description, and relative position in every label record.

If the INPUT, OUTPUT, or I-O option is specified, the USE procedures do not apply respectively to input, output, or input-output files that are described with the LABEL RECORDS ARE OMITTED clause.

The label procedures must not execute any input or output statement.

#### USE BEFORE REPORTING PHRASE

The designated procedures are executed by the Report Writer just before the named report group is produced, regardless of page, overflow, or control break associations with report groups. The report group may be any type except DETAIL.

The indicated identifier represents a nondetail report group named in the Report Section of the Data Division. The identifier must not appear in more than one USE statement.

No Report Writer statement (INITIATE, GENERATE, or TERMINATE) may be written in a procedural paragraph or paragraphs following the USE sentence in the declarative portion.

## FILE PROCESSING EXAMPLES

The following coding describes the sequential accessing of a master file assigned in the SELECT sentence of the FILE-CONTROL paragraph, whose file-code will be associated with a mass storage device on the control card at object program execution.

Example:

```
ENVIRONMENT DIVISION.
FILE-CONTROL.
 SELECT MASTER-FILE ASSIGN TO MF
 ACCESS MODE IS SEQUENTIAL
 PROCESSING MODE IS SEQUENTIAL.
DATA DIVISION.
FILE SECTION.
FD MASTER-FILE LABEL RECORDS ARE STANDARD.
01 M-RECORD.
.
.
PROCEDURE DIVISION.
A. OPEN I-O MASTER-FILE.
B. READ MASTER-FILE AT END GO TO DONE.
. } process M-RECORD
. }
. }
 IF (no change in contents of M-RECORD) GO TO B.
C. WRITE M-RECORD INVALID KEY PERFORM (user routine).
 GO TO B.
DONE. CLOSE MASTER-FILE.
```

A file opened in the I-O state must be read initially and each logical record is either bypassed with no change or else modified and then written back on the storage device. If a WRITE statement is immediately followed by another WRITE statement, the file is positioned to the next logical record before the second WRITE. An ACTUAL KEY phrase may be included for the mass storage file but, in the sequential-access mode, its value does not control the records accessed. The ACTUAL KEY contents are automatically updated, for information only, as the file is sequentially processed through the values 1, 2, 3, ..., until an end-of-file condition is reached. The standard end-of-file mark (octal 17) is then placed in the ACTUAL KEY data-name.

The following coding describes the random accessing in a sequential manner of a master file assigned in the SELECT sentence of the FILE-CONTROL paragraph whose file-code will be associated with a mass storage device on the control card at object program execution.

Example:

```
ENVIRONMENT DIVISION.
FILE-CONTROL.
 SELECT MASTER-FILE ASSIGN TO MF
 FILE-LIMITS ARE 0 THRU 179
 ACCESS MODE IS RANDOM
 PROCESSING MODE IS SEQUENTIAL
 ACTUAL KEY IS ACTUAL-KEY-1.
DATA DIVISION.
FILE SECTION.
FD MASTER-FILE LABEL RECORDS ARE STANDARD
 BLOCK CONTAINS 384 CHARACTERS.
01 M-RECORD.
 02 REC-KEY PIC 9(6).
 02 OTHER-DATA PIC X(378).
WORKING-STORAGE SECTION.
77 ACTUAL-KEY-1 PIC 9(6) COMP-1, VALUE 0.
PROCEDURE DIVISION.
A. OPEN I-O MASTER-FILE.
B. READ MASTER-FILE INVALID KEY GO TO INVALID-ROUTINE.
 .
 . } process M-RECORD
 .
 IF (no change in contents of M-RECORD) ADD 1 TO ACTUAL-KEY-1
 GO TO B.
C. WRITE M-RECORD INVALID KEY GO TO INVALID-ROUTINE.
 ADD 1 TO ACTUAL-KEY-1 GO TO B.
INVALID-ROUTINE.
 IF ACTUAL-KEY-1 IS LESS THAN 180 PERFORM (user routine).
DONE. CLOSE MASTER-FILE.
```

This example presumes that the master file has been assigned three random links on a disk device by means of a file control card. It is this provision and not the FILE-LIMITS phrase in the FILE-CONTROL paragraph that determines the size of the file on the mass storage device. The BLOCK CONTAINS clause in the FD entry establishes a record block size of 64 words which is the logical record size in this example. Then the number of record blocks per link is  $3840/64=60$  or 180 for three links. Since the ACTUAL KEY is incremented by 1, from 0 through 179, over the three links, the INVALID KEY procedures are engaged when the value 180 is reached. If the BLOCK CONTAINS clause were omitted, the maximum record size is used as the block size for each access.

Although the example shows the updating of current records on a random-access file, other possibilities are not precluded. That is, it is possible to insert new records within a file if file space exists within the number of links available. However, deletion of records may only be accomplished by writing over the record space which no longer contains pertinent information.

Even though the operating system does not provide for label records on a random-access mass storage device, it is still necessary to include the LABEL RECORDS ARE STANDARD clause in the FD entry.

## SECTION VI

### LOW-VOLUME DATA TRANSMISSION

#### ACCEPT STATEMENTS

The ACCEPT statements provide access to various low-volume input character data sources.

The specific devices accessed by ACCEPT statements vary considerably among the COBOL compilers of different computers, depending upon the availability of external devices and the characteristics of the operating system. Therefore, the language allows ACCEPT statements to be as machine independent as possible. In an ACCEPT statement, the desired source of the data is specified by a mnemonic-name originated by the user. This mnemonic-name must in turn be associated with a specific data source in the SPECIAL-NAMES paragraph of the Environment Division. Thus, the medium to be used is actually mentioned only in the Environment Division, which is recognized to be highly machine dependent.

The ACCEPT statement permits input access to:

1. GIN (the system input feature of the operating system).
2. COMMUNICATION-DEVICE (the Transaction Processing interface; see Section VII).
3. REMOTE (a terminal not operating under the control of the Transaction Processing System).
4. GLAPS (an operating system feature that provides accumulated processor time for the current run unit).
5. GTIME (an operating system feature that provides the system date and the system clock time).
6. CONSOLE and TYPEWRITER (the system operator interface).
7. SWITCH (a portion of the program switch word, a special software feature provided by the operating system).

When the FROM mnemonic-name phrase is not used in an ACCEPT statement, the input source is considered to be system input (GIN). The FROM phrase must be specified for any other input source. Specific conventions for the various ACCEPT sources are described throughout this section.



## DISPLAY STATEMENTS

The DISPLAY statements provide access to various low-volume output character data destinations.

The specific devices accessed by DISPLAY statements vary considerably among the COBOL compilers of different computers, depending upon the availability of external devices and the characteristics of the operating system. Therefore, the language allows DISPLAY statements to be as machine independent as possible. In a DISPLAY statement, the desired destination of the data is specified with a mnemonic-name originated by the user. This mnemonic-name must in turn be associated with a specific data destination in the SPECIAL-NAMES paragraph of the Environment Division. Thus, the medium to be used is actually mentioned only in the Environment Division, which is recognized to be highly machine dependent.

The DISPLAY statement permits output access to:

1. SYSOUT (the low-volume system output feature of the operating system).
2. COMMUNICATION-DEVICE (the Transaction Processing interface; see Section VII).
3. REMOTE (a terminal not operating under the control of the Transaction Processing System).
4. CONSOLE and TYPEWRITER (the system operator interface).
5. SWITCH (a portion of the program switch word, a special software feature provided by the operating system).

When the UPON mnemonic-name phrase is not used in a DISPLAY statement, the output destination is considered to be system output (SYSOUT). The UPON phrase must be specified for any other output destination. Specific conventions for the various DISPLAY destinations are described throughout this section.

The following rules apply to all DISPLAY statements:

1. When the DISPLAY statement specifies multiple operands, the data characters associated with each operand are concatenated in the order of the occurrence of the operands. Operands are not automatically separated by spaces.
2. The first character of the first operand is positioned in the first character position of a line, subject to the effects of any horizontal and vertical tabulation control characters embedded in the data. If such characters are used, line length limits must not be exceeded.
3. Identifiers must have been described with USAGE DISPLAY (explicitly or implicitly) or DISPLAY-1. Literals may be figurative constants, in which case their size is understood to be one character. ALL has no significance.

## DATA TRANSMISSION TECHNIQUES

### System Input

System input (GIN) is a special card image input provision of the operating system. Along with the control cards submitted to schedule an activity for execution, the user may submit data cards intended as input for the program. Such a collection of data cards may be formally regarded as an input file, in which case the program must include the provisions described for input files in the File Declaration paragraph in Section V. Alternatively, each card may be regarded as an independent item of input data, to be accessed via an ACCEPT statement. The latter approach is applicable when the card format is suitable and the volume is limited.

To utilize GIN via ACCEPT statements, the user may either omit the FROM phrase, or associate a mnemonic-name with GIN in the SPECIAL-NAMES paragraph of the Environment Division. The data item mentioned in the ACCEPT statement must then be a USAGE DISPLAY item. The input item is assumed to occupy the leftmost character positions of the card. No automatic format check or conversion is provided, so it is recommended that the user employ IF statements to assure that the input card contents satisfy the receiving item's description. Similarly, no automatic end-of-file provision is available, so the user must provide an end-of-file test if the volume of system input data can vary. However, each ACCEPT statement executed after the system input is exhausted obtains all spaces, as if a blank card had been read.

The utilization of ACCEPT statements that receive data from GIN in a module overlay environment must be carefully planned to avoid possible overlay loading on top of the COBOL subroutine that controls input from GIN. One method that may be used to avoid such an overlay is to place at least one ACCEPT statement in the main module that will never be overlaid. That statement need not actually be executed.

### System Output

System output (SYSOUT) is a special output collecting provision of the operating system. The object program and system programs can transmit printer line images to SYSOUT, where they are collected on a storage device. After the execution of the object program is terminated, media conversion of the SYSOUT data to the printer is automatically scheduled and accomplished.

Unless overridden by the options specified in the variable field of the \$ LIMITS card, the operating system limits the total volume of data that SYSOUT can receive from a single execute activity to 5000 unit records (corresponding to print lines). In addition, SYSOUT has only limited provision for structuring special preprinted or multiple-part forms on the printer. Subject to these limitations, an activity may transmit as many as eight intermixed reports to SYSOUT, one of which may be the result of DISPLAY statements. For reports having considerable complexity, the Report Writer feature is recommended. For a limited number of lines of miscellaneous information, however, the use of SYSOUT via DISPLAY statements may be more practical.

To utilize SYSOUT via DISPLAY statements, the user may either omit the UPON phrase, or associate a mnemonic-name with SYSOUT in the SPECIAL-NAMES paragraph of the Environment Division. A printer line is considered to contain 132 character positions. The DISPLAY statement may produce more than one line of printing to SYSOUT if the cumulative size of the referenced operands exceeds a total of 132 characters.

To maintain machine independence, printer control characters should not be used in SYSOUT lines. However, hardware dependent slew control can be exercised with the figurative constant HIGH-VALUE which, in a DISPLAY statement, represents a single format control 'escape' character. Thus, the DISPLAY HIGH-VALUE '1' statement slews one line, DISPLAY HIGH-VALUE '2' slews two lines, etc., up to a maximum of nine lines. The DISPLAY HIGH-VALUE SPACE statement causes the printer to slew to the top of the next page.

The utilization of DISPLAY statements that transmit data to SYSOUT in a module overlay environment must be carefully planned to avoid possible overlay loading on top of the COBOL subroutine that controls output for SYSOUT displays. One method that may be used to avoid such an overlay is to place at least one DISPLAY statement in the main module that will never be overlaid. That statement need not actually be executed.

### Transaction Processing Interface

The Transaction Processing System interface is accessed by specifying the ACCEPT MESSAGE and DISPLAY statements using a mnemonic-name that corresponds to the mnemonic-name associated with COMMUNICATION-DEVICE in the SPECIAL-NAMES paragraph. Refer to Section VII for a description of the COBOL and Transaction Processing System interface, and to the Transaction Processing System User's Guide for specific format conventions and programming applications.

### Remote Devices

By specifying a mnemonic-name associated with REMOTE in the SPECIAL-NAMES paragraph of the Environment Division, a COBOL program that interacts with a remote terminal device can be produced. The resulting object program is submitted for execution in the same manner as a program that uses the direct-access file interface. The terminal operator may then request connection to the executing program through the operating system's time sharing or direct-access interfaces. In order to establish interaction with the program's ACCEPT and DISPLAY statements, the inquiry name must be limited to the job number associated with the executing program.

A remote terminal is treated as a type of unit record device. For input purposes, the unit record is assumed to be 80 characters in length. For output purposes, the unit record is assumed to be 72 characters in length.

Each ACCEPT statement whose mnemonic-name is associated with REMOTE will cause a single interaction with the remote terminal. The system operator is notified that a response is expected by a carriage return followed by the display of the character '?' and the ringing of the terminal's bell, if the terminal is so equipped. The input characters, if any, are converted to Hollerith, USAGE DISPLAY characters and are moved into the data item referenced in the ACCEPT statement. The referenced data item should be described, either explicitly or implicitly, as USAGE DISPLAY. No automatic format check or radix conversion is performed.

Each DISPLAY statement whose mnemonic-name is associated with REMOTE will cause from one to four lines to be displayed on the remote terminal, depending upon the size of the referenced data items. Each line will contain at most 72 characters, thereby limiting the total size of the referenced items to 288 characters.

## Elapsed Processor Time

GLAPS is a special operating system feature that provides the processor time accumulated by the current run unit. The increments of time are given in units of 1/64 millisecond.

To utilize GLAPS via ACCEPT statements, GLAPS must be associated with a mnemonic-name in the SPECIAL-NAMES paragraph of the Environment Division, and that name must be referenced in the FROM mnemonic-name phrase of an ACCEPT statement.

To obtain time resolution in units of 1/64 millisecond, the receiving identifier item must be a working-storage data item whose description is equivalent to the following:

```
77 data-name PICTURE 9(10) USAGE COMPUTATIONAL-3.
```

## Date and Time

GTIME is a special operating system feature that provides the current system date and system clock time. The date is given in terms of the month, the day, and the year. If the HMS option is specified for the mnemonic-name definition, the time is given in hours, minutes, and seconds; otherwise, the time is given in units of 1/64 millisecond.

To utilize GTIME via ACCEPT statements, GTIME must be associated with a mnemonic-name in the SPECIAL-NAMES paragraph of the Environment Division, and that name must be referenced in the FROM mnemonic-name phrase of an ACCEPT statement.

To obtain time resolution in units of 1/64 millisecond, the receiving identifier item must be a working-storage data item whose description is equivalent to the following:

```
01 data-name.
02 MONTH PICTURE 99.
02 DAY-OF-MONTH PICTURE 99.
02 YEAR PICTURE 99.
02 TYME PICTURE 9(10) USAGE COMPUTATIONAL-3.
```

If HMS is specified in the GTIME phrase to obtain time resolution in terms of hours, minutes, and seconds, the identifier item must be described differently. The receiving data item for the time must not be described as USAGE COMPUTATIONAL-3 but rather as USAGE DISPLAY (explicitly or implicitly). For example, the receiving identifier could be described as:

```
01 data-name.
02 MONTH PICTURE 99.
02 DAY-OF-MONTH PICTURE 99.
02 YEAR PICTURE 99.
02 TYME PICTURE 9(6).
```

NOTE: The data-names used in these examples are for illustration only.

## Console or Typewriter

Because of the overall demands on the system operator, undisciplined interaction with the operator's console may have an adverse affect on system performance. The use of ACCEPT and DISPLAY statements for system operator console interaction is not recommended in a multiprogramming environment.

The implementor-names TYPEWRITER and CONSOLE are both associated with the system operator console. However, their use implies different types of interaction with that console.

In some cases, communication with the operator either does not require a response from the operator or the response from the operator is not necessarily associated with a previous message sent to the operator. The implementor-name TYPEWRITER is provided for this type of communication with the operator. The use of the implementor-name TYPEWRITER emphasizes the treatment of the console as a unit record device. Successive interactions are treated as unrelated program steps. In the multiprogramming environment, interactions that are caused by two successive statements in the source program may be widely separated by other console interactions initiated by other programs or by the operating system.

Since there are many sources and types of operator messages in a multiprogramming environment, it is usually necessary to ensure that a message that requires a response is closely associated with the request for that response. The implementor-name CONSOLE provides this capability. The use of the implementor-name CONSOLE emphasizes interactive communication with the system operator. Successive DISPLAY and ACCEPT statements, even when widely separated in the source program, are treated as interrelated parts of a single console interaction that occurs when the ACCEPT statement is executed. In this manner, the displayed text is able to immediately precede a related request for input. It is suggested that the DISPLAY statement be used to inform the operator of the nature of the expected response.

Physically, the operator console is treated as a type of unit record device. For input purposes, the unit record is assumed to be 80 characters in length. For output purposes, the record is assumed to be 72 characters in length.

Each ACCEPT statement whose mnemonic-name is associated with TYPEWRITER will cause a single interaction with the operator's console. The system operator is notified that a response is expected by a carriage return followed by the message 'TYPEIN EXPECTED...'. The input characters, if any, are treated as Hollerith, USAGE DISPLAY characters and are moved into the data item referenced in the ACCEPT statement. The referenced data item should be described, explicitly or implicitly, as USAGE DISPLAY. No automatic format check or radix conversion is performed.

Each DISPLAY statement whose mnemonic-name is associated with TYPEWRITER will cause from one to four lines to be displayed on the system console, depending on the size of the referenced data items. Each line will contain at most 72 characters, thereby limiting the total size of the referenced items to 288 characters. If more than one line is emitted for a given DISPLAY statement, there is no assurance, in a multiprogramming environment, that the lines will be juxtaposed on the console display.

Each ACCEPT statement whose mnemonic-name is associated with CONSOLE will cause a single interaction with the operator's console. The system operator is notified that a response is expected by the display of a message. The message will have one of the following forms:

- a. If a DISPLAY statement associated with CONSOLE has been executed, the message will be the text of the last line associated with the latest prior DISPLAY statement. The message will be followed by the characters '???'.
- b. If no prior DISPLAY statement associated with CONSOLE has been executed, the message will be 'TYPEIN EXPECTED...'.

The input characters, if any, are treated as Hollerith, USAGE DISPLAY characters and are moved into the data item referenced by the ACCEPT statement. The referenced data item should be described, explicitly or implicitly, as USAGE DISPLAY. No automatic format check or radix conversion is performed.

Each DISPLAY statement whose mnemonic-name is associated with CONSOLE will allow from one to four lines to be displayed on the system console, depending on the size of the referenced data items. Each line will contain at most 72 characters, thereby limiting the total size of the referenced items to 288 characters. The output line (or, if more than one line results from the statement, the last output line) is held in a buffer until the next execution of an ACCEPT statement associated with CONSOLE. At that time, the line will be used to inform the operator of a pending need for a response.

The ACCEPT and DISPLAY statements need not appear together in the source program, provided that the DISPLAY statement is executed first. Should no ACCEPT statement be executed after the DISPLAY statement, the output data is not displayed. Two DISPLAY statements of this kind with no intervening ACCEPT statement would result in the suppression of the output from the first DISPLAY statement. If more than one line results from an execution of a DISPLAY statement associated with CONSOLE, all lines except the last line are emitted at once and, in a multiprogramming environment, there is no assurance that any of the lines will be juxtaposed on the console display.

### Switches

The operating system establishes a 'program switch word' for each activity. The 36 bits of each program switch word represent 36 software 'sense switches'. For COBOL purposes, the switches are numbered 0, 1, 2, ..., 35. Switches 0-5 and 12-17 are reserved for the use of the operating system. Switches 18-35 may be used for communicating between activities. Switches 6-11 are reset at the beginning of each activity using the ON option of the \$ EXECUTE card. (Refer to the Control Cards reference manual.) Each of the switches 6-11 is set OFF at the beginning of each activity unless an option of the \$ EXECUTE card causes it to be set ON, as shown below:

| <u>ON Setting<br/>for Switch</u> | <u>\$ EXECUTE<br/>Card Option</u> |
|----------------------------------|-----------------------------------|
| 6                                | ON1                               |
| 7                                | ON2                               |
| 8                                | ON3                               |
| 9                                | ON4                               |
| 10                               | ON5                               |
| 11                               | ON6                               |

A switch is ON if its value is one (1) and OFF if its value is zero (0).

Each switch to be used must be associated with a mnemonic-name in the SPECIAL-NAMES paragraph of the Environment Division. In the Procedure Division, an ACCEPT...FROM statement may cause a data item to receive the switch setting, or a DISPLAY...UPON statement may cause the switch to be set ON or OFF according to the following rules:

1. Only one operand is permitted in a DISPLAY statement that transmits data to the switches; that is, if a DISPLAY statement refers to a mnemonic-name associated with a switch, only one operand (data-name, literal, or the figurative constant ZERO) may be given. If the value of the operand is 1, the switch will be set ON; if the value is 0, the switch will be set OFF. If a literal is used, it must be an integer that has a value of 1 or 0. If a data-name is specified, it must be a COMPUTATIONAL-1 data item in the Working-Storage Section, with a size not exceeding eight digits. The following data description is recommended:

```
77 data-name PICTURE 9 COMPUTATIONAL-1.
```

If the value of the item exceeds one (1), the value modulo 2 determines the switch setting.

2. If a mnemonic-name associated with SWITCH is specified, the ACCEPT statement causes the value of data-name to be set to 1 if the switch is ON, or set to 0 if the switch is OFF. The data-name must be a data item in the Working-Storage Section whose description is equivalent to the following:

```
77 data-name PICTURE 9 COMPUTATIONAL-1.
```

An alternative method of testing a switch is provided by the switch-status condition test (see Section VII in the COBOL Reference Manual). In this application, a mnemonic-name must be specifically associated with either the ON or OFF setting of a switch.

To illustrate the use of switches, consider that one program must determine whether or not a subsequent program is to produce a certain report. The report is to be produced only if switch 35 is set ON. The setting of the switch can be programmed as follows:

Program 1 (Setting switches):

SPECIAL-NAMES phrase:

```
SWITCH 35 IS REPORT-CONTROL
 { ON STATUS IS ... }
 { OFF STATUS IS... }
```

Procedure Division statement:

```
IF ... DISPLAY 1 ON REPORT-CONTROL
ELSE DISPLAY 0 ON REPORT-CONTROL.
```

The decision to produce the report or not can be programmed in one of the following two ways:

Program 2 (Sensing switches):

SPECIAL-NAMES phrase:

SWITCH 35 IS REPORT-CONTROL  
ON STATUS IS  
DO-REPORT.

Procedure Division statement:

IF DO-REPORT  
PERFORM ...

SPECIAL-NAMES phrase:

SWITCH 35 IS REPORT-CONTROL  
ON STATUS IS  
DO-REPORT.

Data Division syntax:

77 SWITCH-VALUE PIC 9 COMP-1.

Procedure Division statements:

ACCEPT SWITCH-VALUE FROM  
REPORT-CONTROL.  
IF SWITCH-VALUE = 1  
PERFORM ...





DATA TRANSMISSION PROGRAM EXAMPLE

An example of a conversational COBOL program is presented below. The program is entered as a CARDIN job from a remote terminal. (Refer to the TSS Terminal/Batch Interface manual for details concerning CARDIN.) The program is caused to first compile and then execute with the user's terminal connection to the batch job by a request to RUN followed by the operator responses (underlined in the example).

Example:

```
10$;IDENT;VHA73,STATION-F
20$;COBOL;NDECK,LSTOU,ON6
30;IDENTIFICATION DIVISION.
40;PROGRAM-ID. TALK.
50;ENVIRONMENT DIVISION.
60;CONFIGURATION SECTION.
70;SOURCE-COMPUTER. 6000 WITH EIS.
80;OBJECT-COMPUTER. 6000 WITH EIS.
90;SPECIAL-NAMES.
100;;REMOTE IS REM.
150;FILE-CONTROL.
160;I-O-CONTROL.
200;DATA DIVISION.
210;FILE SECTION.
300;WORKING-STORAGE SECTION.
310;77 MESSAGE-RQST PIC 9 COMP-1.
320;77 END-RQST PIC X(3).
1000;PROCEDURE DIVISION.
1010;START SECTION.
1020;TELL-ON-THE-AIR.
1030;;DISPLAY 'START OF EXAMPLE TALK RUN' UPON REM.
1040;;DISPLAY 'INPUT CHOICE OF TASK BY TYPING 001,002, OR 003'
1045;; UPON REM.
1050;TAKE-MESS. ACCEPT END-RQST FROM REM.
1060;;IF END-RQST = 'END' GO TO DONE.
1065;;MOVE END-RQST TO MESSAGE-RQST.
1070;;GO TO TASK1, TASK2, TASK3
1080;; DEPENDING ON MESSAGE-RQST.
1090;;DISPLAY 'BAD REQUEST--RETRY OR TYPE END' UPON REM.
1100;;GO TO TAKE-MESS.
1110;TASK1.
1120;;DISPLAY 'TASK1 IS COMPLETED' UPON REM.
1130;;GO TO ASK-FOR-MORE.
1140;TASK2.
1150;;DISPLAY 'TASK2 IS COMPLETED' UPON REM.
1160;;GO TO ASK-FOR-MORE.
1170;TASK3.
1180;;DISPLAY 'TASK3 IS COMPLETED' UPON REM.
1190;ASK-FOR-MORE.
1200;;DISPLAY 'READY FOR NEXT--TYPE 001,002,003 OR END'
1210;; UPON REM GO TO TAKE-MESS.
1220;DONE. DISPLAY 'TALK RUN COMPLETED' UPON REM.
1230;;STOP RUN.
9000$;EXECUTE;DUMP
9009$;ENDJOB
```

READY

Example (cont):

```
*RUN
 SNUMB # 5532T
CARD FORMAT,DISPOSITION ?
M,T
TAB CHARACTER AND SETTINGS?
; , 8, 16
START OF EXAMPLE TALK RUN
INPUT CHOICE OF TASK BY TYPING 001,002, OR 003
?001
TASK1 IS COMPLETED
READY FOR NEXT--TYPE 001,002,003 OR END
?002
TASK2 IS COMPLETED
READY FOR NEXT--TYPE 001,002,003 OR END
?003
TASK3 IS COMPLETED
READY FOR NEXT--TYPE 001,002,003 OR END
?999
BAD REQUEST--RETRY OR TYPE END
?END
TALK RUN COMPLETED
ACTIVITY TERMINATED
*NORMAL TERMINATION
```

## SECTION VII

### TRANSACTION PROCESSING SYSTEM

The Transaction Processing System (TPS) consists of the Transaction Processing Executive (TPE) that is part of the operating system, and the Transaction Processing Applications Programs (TPAPs) that are written by the user to process transactions.

Some of the features of the Transaction Processing Executive are discussed in this section in general terms to assist the user to understand the interface between the user-supplied TPAP and the Transaction Processing System. However, to obtain specific format conventions and programming details, refer to the Transaction Processing System User's Guide.

#### TRANSACTION PROCESSING EXECUTIVE (TPE)

The Transaction Processing Executive controls the receipt of transactions from terminals and delivers the transactions to appropriate Transaction Processing Applications Programs for processing. The TPE also directs output messages to terminals from the TPAPs. If a requested terminal is not accessible, the TPE holds the output for later transmission.

If direct communication between a user at a terminal and a TPAP is required, the TPE schedules the TPAP and performs the line switching necessary for direct-access communication.

#### TPAP Profile Table

The user must provide the Transaction Processing Executive with information that will enable the TPE to associate an input message with the appropriate TPAP. This information is assembled into the TPE using a macro.

The user-supplied parameters for the macro that are applicable to a COBOL TPAP are:

1. A unique three-character TPAP identifier (ID) that matches the first three characters of the program-name in the TPAP's PROGRAM-ID paragraph.
2. Nine binary digits (bits), used to indicate whether or not:
  - a. The TPAP is to be called into execution with the special message \*\*\*STRT and possibly terminated with the special message \*\*\*TERM.

- b. The TPAP input is in BCD format (ASCII format is not supported by COBOL).
  - c. The TPAP output is in BCD format.
  - d. Message input order is to be maintained.
  - e. A line switch is required (direct-access mode).
  - f. A journal of the transaction messages is required.
  - g. The TPAP is to remain in memory and be available.
3. The TPAP input buffer size.
  4. The TPAP output buffer size (assigned in the TPE).
  5. The type of buffer assignment; fixed or dynamic.
  6. The priority of the TPAP (optional) to indicate queueing order.
  7. A list of keywords and their associated priorities (optional).
  8. The number of keywords.

Transaction Message Format

The format of the transaction message accepted by the TPE is flexible. The only required field in the message is the keyword that the TPE uses to identify and schedule the appropriate TPAP. The format of the transaction message is:

| <u>Field 1</u> | <u>Field 2</u> | <u>Field 3</u> |
|----------------|----------------|----------------|
| (xxx)          | kkkkkkkk       | mm...m         |

Field 1 (xxx) is the optional logical terminal identifier (ID). This logical ID is composed of three alphanumeric characters enclosed in parentheses. It is used primarily to identify the terminal to the TPAP for directing the transaction output. If the logical ID is not specified, the physical line ID is used. However, the physical line ID is only valid while the terminal is connected. If the terminal is disconnected before the output is delivered, there will be no valid destination ID, and the output will be undeliverable.

Field 2 (kkkkkkkk) describes the transaction keyword. The keyword may be from one to eight characters in length and must be followed by a blank or comma if it is less than eight characters. The keyword identifies and, in effect, selects the TPAP to be used to process the transaction. A TPAP may have one or more keywords; these keywords may carry priorities in order to establish the queueing priority structure in the TPE. If the keyword in the incoming transaction message cannot be matched with a keyword associated with one of the TPAPs, the transaction input is rejected. An entry for an acceptable transaction request is placed in a queue depending on the priority associated with the keyword and the TPAP.

Field 3 (mm...m) contains the message text. The format of this message is determined by the TPAP. The message must be terminated with a special end-of-message symbol defined by the user installation. The message length must meet the requirements for the type of terminal being utilized.

## TRANSACTION PROCESSING APPLICATIONS PROGRAMS (TPAPS)

The TPAPs are user-supplied programs that are written to process any number of different functions. These functions can be differentiated by keywords (TPAP identifiers). Any number of keywords can be associated with a TPAP.

Since the applications programs are designed and developed by the user, the functions performed by the TPAP are at the discretion of the user.

### TPE/TPAP Interface

The TPE and the TPAP do not communicate directly with one another. All communication between the two modules is accomplished through the intercom input and output files.

If the TPAP is written in COBOL, the compiler provides the necessary interface for communicating with the TPE using the COMMI and COMMO subroutines. The communication of information between the TPE and each TPAP is through a buffer-to-buffer exchange of data.

The COBOL PROGRAM-ID for a TPAP must contain a unique program-name in the first three characters of the maximum six-character name. The program-name is used to identify a given TPAP to the system, and is the name associated with the keywords used in the transaction input. The standard method of generating file-names to ensure a linkage between the files of the TPE and the intercom files of the TPAP that are generated by the COBOL compiler is described below:

1. The input file (called the intercom input file) is created with the name ZxxxI, in which xxx represents the first three characters of the program-name specified in the COBOL PROGRAM-ID paragraph. The TPAP receives input data via this file from a correspondingly named file in the TPE.
2. The output file (called the intercom output file) is created with the name ZxxxO. The TPAP will transmit data from this file to a file in the TPE that has a corresponding name.

### Intercom Input File Processing

Each input to the TPAP via the intercom input (intercom I) file contains the entire text of the message from the terminal plus an input header. The length of the text is limited to the length that the Transaction Processing Executive is capable of processing and to the defined buffer sizes. The input message may not be segmented.

The input header description has the following implied COBOL record description:

```
01 INPUT-HDR.
 02 INPUT-TRANS-NO PICTURE 9(8) COMP-1.
 02 INPUT-STATUS PICTURE 9(8) COMP-1.
 02 INPUT-SIZE PICTURE 9(8) COMP-1.
 02 SOURCE-ID PICTURE X(3) SYNC LEFT.
 02 MESSAGE-DATE PICTURE 9(6).
 02 MESSAGE-TIME PICTURE 9(8) COMP-1.
 02 QUEUE-DEPTH PICTURE 9(8) COMP-1.
```

These implicit descriptions provide access to data pertaining to the input messages. The user can specify these names (except INPUT-HDR) for other purposes within the program as long as each reference within the program (including any reference to the elementary items within the input header) is qualified.

For example, if the TPAP contains a transit number table described as:

```
01 TRANSIT.
 02 INPUT-TRANS-NO OCCURS 10 TIMES
 INDEXED BY INPUT-TRANS-NDX.
 03 DISTRICT-1 PIC 9(4) DISPLAY.
 .
 .
 03 DISTRICT-10 PIC 9(4) DISPLAY.
```

The user must specify INPUT-TRANS-NO of TRANSIT (INPUT-TRANS-NDX) to access the appropriate transit number within the table.

The user must specify INPUT-TRANS-NO of INPUT-HDR to access the transaction number within the message input header.

#### INPUT SUBROUTINE (COMMI)

The input subroutine (COMMI) is called for an ACCEPT MESSAGE statement that implies input from a terminal.

The three-character PROGRAM-ID is used by the input subroutine to form a file-name (ZxxxI) needed to communicate with the intercom file. The input subroutine contains one 210-word buffer (ZINBUF) that is of sufficient size to hold a full screen of data (26 lines) from a keyboard display terminal. (The TPE always sends complete messages, that may be up to a full screen in size.) The first nine words of the buffer contain the message header after input; message text always begins in the tenth word. The COBOL compiler generates the implicit record description to describe this input header contained in the transaction request.

After issuing a read to the intercom file, the COMMI delays for a status condition on the read. If data is available, it is moved to the internal work area specified in the ACCEPT MESSAGE statement. The message is transferred to the receiving area (referenced by identifier) left-justified without space-fill.

There is no end-of-segment indicator, since an end-of-message indicator is included with each transmission from the TPE.

## COBOL INPUT STATEMENT PROCESSING

The user need not reference the input header directly. The appropriate references will be made automatically by the COBOL compiler when the ACCEPT MESSAGE identifier FROM mnemonic-name statement is specified. Identifier is the name of the record area within the TPAP program that is to receive the transaction message (excluding the input header information). The identifier must be a level 01 or a level 77 data item in the Working-Storage Section.

The mnemonic-name in the ACCEPT MESSAGE statement must be associated with the mnemonic-name specified in the COMMUNICATION-DEVICE phrase of the SPECIAL-NAMES paragraph. The compiler generates the implicit input and output header descriptions and places them into the data-name table when the COMMUNICATION-DEVICE phrase is processed.

When no data is ready to be processed by the TPAP, the statements to be executed can be specified by including the NO DATA imperative-statement phrase with the ACCEPT MESSAGE statement. The input subroutine COMMI provides the NO DATA return. The specified imperative-statement is executed when no data is available.

If the NO DATA phrase is omitted from a TPAP, control is returned to the statement immediately following the ACCEPT MESSAGE statement when data is received; however, if no data is available, the program is suspended until data is available. At that time control is returned to the statement following the ACCEPT MESSAGE statement.

### Intercom Output File Processing

There is no limit to the number of DISPLAY statements a TPAP may issue to send a complete transaction message, although each complete message should not exceed the limit imposed by the type of terminal that is to receive the message. The Transaction Processing Executive accepts only 128 words at a time; therefore, long messages must be segmented by the Transaction Processing Applications Program. Each segment must contain the output header.

The output header description has the following implied COBOL record description entry:

```
01 OUTPUT-HDR.
 02 OUTPUT-TRANS-NO PICTURE 9(8) COMP-1.
 02 OUTPUT-STATUS PICTURE 9(8) COMP-1.
 02 OUTPUT-SIZE PICTURE 9(4) COMP-1.
 02 DEST-COUNT PICTURE 9(4) COMP-1.
 02 DESTINATION PICTURE X(3) SYNC LEFT
 OCCURS 12 TIMES.
```



These implicit descriptions provide access to information which is used to build an output header for a message prior to releasing it to the TPE. These names (except OUTPUT-HDR) may be used for other purposes within a program as long as each reference within the program (including any reference to the elementary items within the output header) is qualified. The program at the end of this section includes an example of the multiple use of DESTINATION within the program for purposes other than the output header. DESTINATION of OUTPUT-HDR (subscript) must be specified to reference the appropriate terminal destination in the message output header.

When the output message is to be transmitted to the same logical identifier and transaction number as the input message, the compiler will automatically initialize the output header description for the TPAP.

When the output message is to be transmitted either to a destination other than that of the input message or to more than one destination, the user must perform the following steps prior to the execution of each DISPLAY statement:

1. Move the transaction number to the output transaction number (OUTPUT-TRANS-NO) if it differs from the input transaction number (INPUT-TRANS-NO).
2. Indicate the number of terminals that are to receive the output message by moving a value from one (1) to twelve (12), inclusive, to the DEST-COUNT field.
3. Move the logical identifier for each receiving terminal into DESTINATION(x), where x is a literal or subscript containing a value from one (1) to twelve (12), inclusive, but not exceeding the value in DEST-COUNT.

The length of the message to be transmitted will be moved automatically by the COBOL object program to the output message size (OUTPUT-SIZE) field regardless of the destination(s) of the message unless it is user specified. In this case, if a value larger than the data field is specified, a CX abort is generated.

#### OUTPUT SUBROUTINE (COMMO)

The output subroutine (COMMO) is called to transmit output (processed transaction data) from the TPAP to the TPE.

The output subroutine in COBOL provides the following capabilities:

1. When a DISPLAY to a communication device is executed, the subroutine first moves the output header information to the intercom 0 file in the TPE, and then moves the message text to that file. The move of the message text is controlled by the number of characters specified in the OUTPUT-SIZE field.
2. Each message moved to the intercom 0 file is prefaced by a word containing a character count which specifies the length of the message.
3. If the OUTPUT-SIZE field contains zeros, the size of the sending field containing the data will be used.

4. If the contents of the OUTPUT-SIZE field contain a message larger than the message that can be contained in the sending field, a CX abort is generated. This field is zeroed after usage by COMMO.
5. If the OUTPUT-STATUS field contains zero, the value generated in the CALL is inserted.
6. If the contents of the OUTPUT-STATUS field are nonzero, then this field will not be modified. This field is zeroed after usage by COMMO.
7. The subroutine generates an END-OF-SEGMENT, END-OF-MESSAGE, or END-OF-TRANSACTION, depending on the contents of the entry to the subroutine.

#### COBOL OUTPUT STATEMENT PROCESSING

The output transaction number in the output header must be initialized by the TPAP prior to the execution of each DISPLAY statement if it is different from the input transaction number in the input header.

The END-OF-SEGMENT, -MESSAGE, -TRANSACTION options may only be used when data is to be displayed upon a communication device specified by mnemonic-name. One of these three modes and a COMMUNICATION-DEVICE (mnemonic-name) must be used to display data to the TPE, regardless of the destination IDs in the header.

When a TPAP is ready to transmit the text of a segment within a long message, the DISPLAY literal (identifier) ... END-OF-SEGMENT UPON mnemonic-name statement should be specified. ESI is an appropriate abbreviation for END-OF-SEGMENT. The ESI or END-OF-SEGMENT option is to be used when the Transaction Processing Executive buffer is too small to contain the entire message.

The mnemonic-name in the UPON phrase must correspond to the mnemonic-name specified in the COMMUNICATION-DEVICE phrase of the SPECIAL-NAMES paragraph. The compiler generates the implicit input and output header descriptions and places them into the data-name table when the COMMUNICATION-DEVICE phrase is processed.

When a TPAP is ready to transmit the text of the last segment of a long message, or all of the text of a short message, the DISPLAY literal (identifier) ... END-OF-MESSAGE UPON mnemonic-name statement should be specified. EMI is an appropriate abbreviation for END-OF-MESSAGE. The EMI or END-OF-MESSAGE option (whether included in the DISPLAY statement or generated by the COBOL compiler) indicates that more messages may follow using the same transaction number. If EMI or END-OF-MESSAGE is not included in the DISPLAY statement and mnemonic-name corresponds to the mnemonic-name specified in the COMMUNICATION-DEVICE phrase of the SPECIAL-NAMES paragraph, the COBOL compiler will automatically transmit an END-OF-MESSAGE symbol at the end of the transaction message.

When a TPAP is ready to indicate that the message text is the end of the last message for the specified transaction number, the DISPLAY literal (identifier) ... END-OF-TRANSACTION UPON mnemonic-name statement should be specified. ETI is an appropriate abbreviation for END-OF-TRANSACTION. It is required to specify a DISPLAY ... END-OF-TRANSACTION statement in order to begin processing another transaction. The TPAP must send an ETI to remove a transaction from execution; the ETI is not automatically generated in the next ACCEPT MESSAGE statement.

When data is to be transmitted to a terminal, the destination ID in the output header (DESTINATION) is used as the logical terminal ID by the TPE. DEST-COUNT will be set to one (1) and DESTINATION (1) will be initialized to the input identifier (SOURCE-ID) if the TPAP does not initialize these fields prior to the execution of each DISPLAY statement.

One TPAP may call another TPAP into execution by specifying the COBOL statements MOVE 1 TO DEST-COUNT. MOVE '\*\*\*' TO DESTINATION (1). , followed by one of the DISPLAY statement options ESI, EMI, or ETI. The output of the first TPAP will be used as input to the second TPAP. The format of the message must be that of the input file format from the communication device to the first TPAP. The keyword in the message identifies the second TPAP that is to be called into execution.

An example of the method used to call another TPAP into execution follows:

Example:

```

.
.
SPECIAL-NAMES.
 COMMUNICATION-DEVICE IS TPS-TERMINAL.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 TPAP-SPAWN.
 02 LOGICAL-ID.
 03 FILLER PIC X VALUE '('.
 03 TERMINAL-ID PIC XXX.
 03 FILLER PIC X VALUE ')'.
 02 TPAP-KEYWORD PIC X(8) VALUE 'ACCT0014'.
 02 TPAP-MESSAGE PIC X(80).
PROCEDURE DIVISION.
PAR-NAME.
 ACCEPT MESSAGE TPAP-SPAWN UPON TPS-TERMINAL.
 MOVE 1 TO DEST-COUNT.
 MOVE '***' TO DESTINATION(1).
 MOVE SOURCE-ID TO TERMINAL-ID.
 DISPLAY TPAP-SPAWN ETI UPON TPS-TERMINAL.
```

#### Direct-Access (DAC) Mode Processing

The COBOL interface may also be used for direct-access (DAC) transaction processing. A TPAP may contain ACCEPT MESSAGE and DISPLAY statements that reference mnemonic-names used in the DAC processing mode. DAC mode processing is specified with the REMOTE IS mnemonic-name phrase in the SPECIAL-NAMES paragraph.

When a TPAP requires direct interactive communication with a terminal, a line switching bit must be set in the TPAP profile table in the TPE. When the line switch is made, the line is logically disconnected from the TPE and a direct connection is established between the terminal and the TPAP. The TPAP can be in direct communication only with the originating terminal. At least one ACCEPT MESSAGE statement must be executed before the TPAP requests a line switch. When the TPAP is ready to accept a message directly from the terminal, the line switch will be established by the TPE when the ACCEPT literal (identifier) ... FROM mnemonic-name statement is specified.

In the DAC mode, a program may accept or display any number of messages to or from the remote device. When the program is ready to accept another transaction from the TPE, it transmits an ACCEPT MESSAGE statement. The interface subroutine recognizes that the last message from the TPAP was in the DAC mode and performs a line switch to return the line to the TPE. When the terminal is returned to the TPE, the previous logical ID(s) of the terminal must be redefined before they can receive output. When the line is switched, the next transaction request is received, and the TPE again switches the line, if requested, to the TPAP so that it can process the transaction in the direct-access mode.

When the TPAP is ready to transmit a message directly to the terminal, the DISPLAY literal (identifier) ... UPON mnemonic-name statement should be specified.

The direct-access transaction operation ends when the TPAP disconnects the line or switches it back to the TPE.

A Transaction Processing Applications Program executing in the direct-access mode must not terminate without sending an END-OF-TRANSACTION message for the transaction being processed.

The Transaction Processing Executive must receive an END-OF-TRANSACTION message for each transaction before it will release the TPAP for processing subsequent transactions.

#### Transaction Processing Applications Program Example

The following program is a simplified version of a TPAP, written in COBOL, that could be used for airline scheduling. In addition to the TPAP, the user would have to supply a data base for the TPAP to apply against the transaction.

000010 IDENTIFICATION DIVISION.  
000020 PROGRAM-ID. XXX.  
000030 AUTHOR.  
000040 DATE-WRITTEN. SEPTEMBER 1974.  
000050 ENVIRONMENT DIVISION.  
000060 CONFIGURATION SECTION.  
000070 SOURCE-COMPUTER. 6000 WITH EIS.  
000080 OBJECT-COMPUTER. 6000 WITH EIS.  
000085 SPECIAL-NAMES.  
000090 COMMUNICATION-DEVICE IS TERM-3.  
000110 FILE-CONTROL.  
000120 SELECT SCHEDULE ASSIGN TO CR FOR CARDS.  
000200 DATA DIVISION.  
000210 FILE SECTION.  
000220 FD SCHEDULE;  
000230 LABEL RECORDS ARE STANDARD;  
000240 DATA RECORDS ARE FLIGHT-SCHED.  
000250 01 FLIGHT-SCHED.  
000260 02 FLIGHT; PICTURE X(4).  
000270 02 FILLER; PICTURE AAAA.  
000280 02 DESTINATION; PICTURE X(20).  
000290 02 FILLER; PICTURE AA.  
000300 02 AIRMILES; PICTURE X(5).  
000310 02 FILLER; PICTURE AAA.  
000320 02 DEPTIME; PICTURE X(8).  
000330 02 FILLER; PICTURE AAA.  
000340 02 ARRIVE; PICTURE X(8).  
000350 02 FILLER; PICTURE X(27).  
000500 WORKING-STORAGE SECTION.  
000501 77 N; PICTURE 99 COMPUTATIONAL-1.  
000502 77 I; PICTURE 99 COMPUTATIONAL-1.  
000503 77 SW; PICTURE 9.  
000504 01 INPUT-REQUEST.  
000505 02 NME PIC XXX.  
000506 02 CMA PIC X.  
000507 02 CHECK-REQUEST PIC X(20).  
000510 01 END-STATEMENT.  
000520 02 JOBDONE; PICTURE A(20) VALUE "DONE".  
000530 01 DATA-SPACE.  
000540 02 DATA-FILE; OCCURS 20 TIMES.  
000550 03 FLIGHT; PICTURE X(4).  
000560 03 DESTINATION; PICTURE X(20).  
000570 03 AIRMILES; PICTURE X(5).  
000580 03 DEPTIME; PICTURE X(8).  
000590 03 ARRIVE; PICTURE X(8).  
000600 01 TEMP-FILE.  
000610 02 FLIGHT; PICTURE X(4).  
000620 02 DESTINATION; PICTURE X(20).  
000630 02 AIRMILES; PICTURE X(5).  
000640 02 DEPTIME; PICTURE X(8).  
000642 02 ARRIVE; PICTURE X(8).  
000700 PROCEDURE DIVISION.  
000710 BEGIN.  
000720 OPEN INPUT SCHEDULE.  
000726 STORER.  
000730 PERFORM DATA-STORAGE VARYING N FROM 1 BY 1  
000740 UNTIL N GREATER THAN 20.  
000748 GO TO PROCESS.  
000750 PROCESSA.  
000751 ENTER LINKAGE MODE.  
000752 POPUP DATA-STORAGE.  
000753 ENTER COBOL.

```
000754 PROCESS.
000755 MOVE SPACES TO INPUT-REQUEST.
000770 ACCEPT MESSAGE INPUT-REQUEST FROM TERM-3 NO DATA
 GO TO WAIT.
000775 EXAMINE CHECK-REQUEST REPLACING ALL ZEROS BY SPACES.
000790 IF CHECK-REQUEST = JOBDONE GO TO ENDD.
000795 MOVE 0 TO SW.
000800 PERFORM DATA-COMPARE VARYING N FROM 1 BY 1 UNTIL
000801 N GREATER THAN 20 OR SW = 1.
000814 IF SW = 1 GO TO WRITER.
000815 DISPLAY "NO COMPARE ON CHECK-REQUEST END-OF-TRANSACTION"
000816 UPON TERM-3.
000820 GO TO PROCESS.
000830 ENDD.
000840 CLOSE SCHEDULE.
000850 STOP RUN.
000900 DATA-STORAGE.
000910 READ SCHEDULE; AT END GO TO PROCESSA.
000920 MOVE CORRESPONDING FLIGHT-SCHED TO TEMP-FILE.
000930 MOVE TEMP-FILE TO DATA-FILE (N).
000940 DATA-COMPARE.
000950 IF CHECK-REQUEST = DESTINATION IN
000960 DATA-FILE (N) MOVE 1 TO SW.
000962 WAIT.
000963 ENTER GMAP.
000964 MME GERELC
000965 ENTER COBOL.
000966 GO TO PROCESS.
000970 WRITER.
000980 DISPLAY "FLIGHT# DESTINATION # AIR-MILES DE
000990- "PT TIME AR-TIME "
001010 FLIGHT IN DATA-FILE (N)," ",DESTINATION IN
001025 DATA-FILE (N)," ",AIRMILES IN DATA-FILE (N)," ",DEP
001026- TIME IN DATA-FILE (N)," ",ARRIVE IN DATA-FILE (N)
001030 "END-OF-TRANSACTION" UPON TERM-3.
001035 GO TO PROCESS.
001200 END PROGRAM.
$ ENDJOB
***EOF
```



## SECTION VIII

### REPORT WRITER

#### DESCRIPTION OF THE REPORT WRITER

The Report Writer provides the facility for producing reports by specifying the physical appearance of a report instead of specifying the detailed procedures necessary to produce the report.

A hierarchy of levels is used to define the logical organization of a report. Each report is divided into report groups, which in turn are divided into sequences of items. This hierarchical structure permits explicit reference to a report group with implicit reference to other levels in the hierarchy. A report group contains one or more items to be presented on one or more lines.

The Report Writer feature places emphasis on the organization, format, and contents of an output report. Although a report can be produced using the standard COBOL language, the Report Writer language characteristics provide a more concise method for report structuring and report production. Much of the Procedure Division coding which would normally be supplied by the user is instead provided automatically by the Report Writer Control System. Thus, the user is relieved of writing procedures for moving data, constructing print lines, counting lines on a page, numbering pages, producing heading and footing lines, recognizing the end of logical data subdivisions, updating sum counters, etc. All of these operations are accomplished by the Report Writer Control System from source language statements that appear primarily in the Report Section of the Data Division of the source program.

Data movement to a report is directed by the Report Section clauses SOURCE, SUM, and VALUE. Fields of data are positioned on a print line by means of the COLUMN NUMBER clause. The PAGE clause specifies the length of the page, the size of the heading and footing areas, and the size of the area in which the detail lines will appear. Data items may be specified to form a control hierarchy. During the execution of a GENERATE statement, the Report Writer Control System uses the control hierarchy to check automatically for control breaks. When a control break occurs, summary information (subtotals) can be presented.

#### Report Format

A report may consist of any meaningful combination of the following syntax selections:

- REPORT HEADING (one for each report)
- PAGE HEADING (one format for each report)



- OVERFLOW HEADING (one format for each report)
- CONTROL HEADING (one format for each control level)
- DETAIL (no limit for each report)
- CONTROL FOOTING (one format for each control level)
- OVERFLOW FOOTING (one format for each report)
- PAGE FOOTING (one format for each report)
- REPORT FOOTING (one for each report)

In COBOL, each report is described in the Report Section of the Data Division. The user specifies the intended format of each of the headings, footings, and detail lines in the report, as well as all sources of data. A report may utilize data described in the File Section and Working-Storage Section. In addition, the user specifies the overall organization and intended page layout of the report.

The compiler provides the following functions in the object program:

1. Vertical format control, including line counting, page counting, and production of page headings and footings.
2. Detection of control breaks.
3. Production of control headings and footings.
4. Accumulation of control totals to any number of control levels.
5. Execution of user-defined procedures before presentation of control headings and footings.
6. Production of overflow headings and footings.

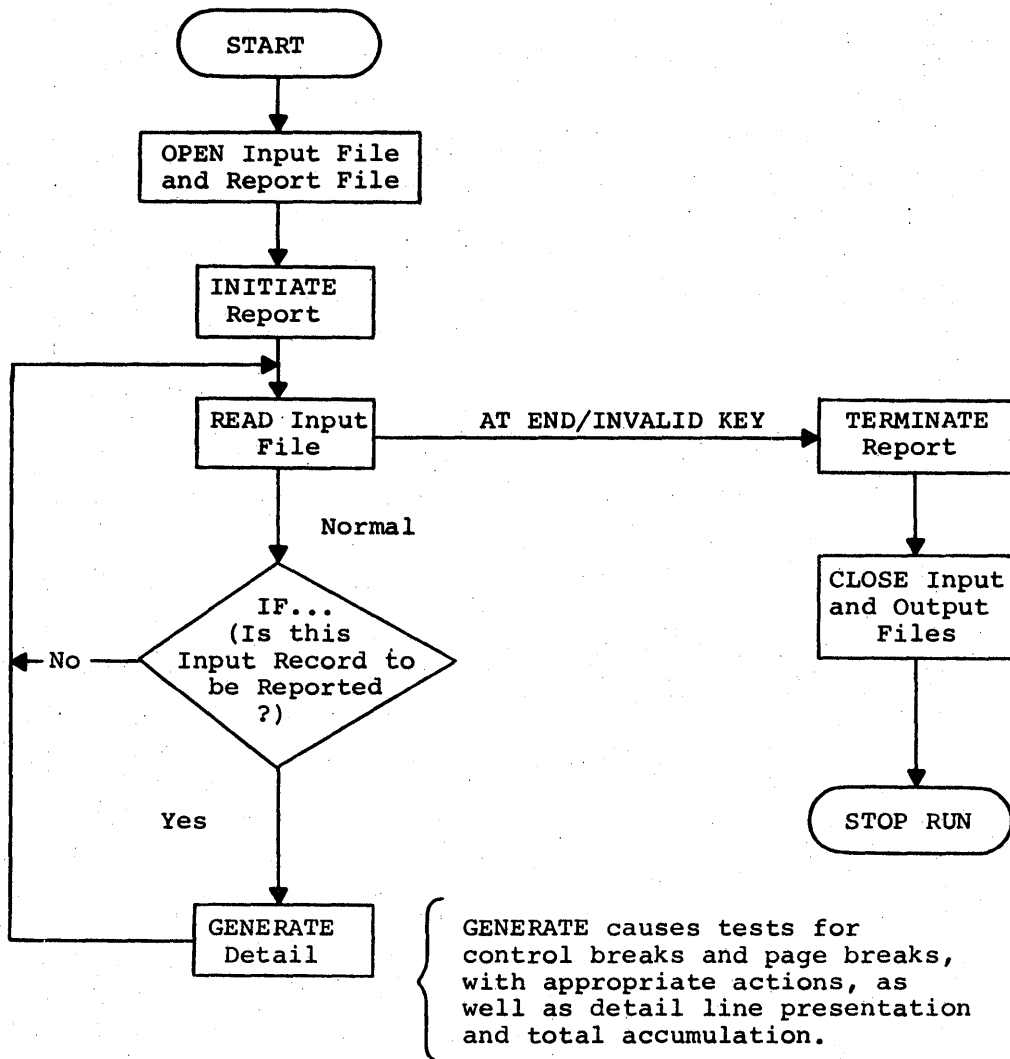
#### Report Control in the Procedure Division

The production of a report is controlled in the Procedure Division with three report writing statements:

- INITIATE
- GENERATE
- TERMINATE

The BEFORE REPORTING phrase of the USE statement may also be used to control the production of a report.

The relationship of the above statements to other Procedure Division statements is illustrated by the following flow chart of a simple reporting program:



Before a GENERATE statement is executed, the report must be initiated. The INITIATE statement causes initial housekeeping values to be established and report and page headings to be presented.

The GENERATE statement provides for all aspects of report editing, writing, and housekeeping, but GENERATE in itself makes no provision for reading input data or deciding when detail lines should be produced. Instead, the user explicitly obtains each input record via COBOL statements such as the READ statement.

When the last GENERATE statement has been executed, the report must be terminated. The TERMINATE statement causes final control footings and report footings to be presented.

The immediate destination of a report is always a file specified in the File Section of the Data Division. The file must be explicitly opened prior to execution of the report's INITIATE statement, and the file must be explicitly closed after the TERMINATE. The report writing statements implicitly perform whatever writing is required for the report.

Skeletal Format for the Report Section

The definition of each report includes two types of entries:

1. The RD entry specifies the basic page layout and the overall organization of the report.
2. Report item description entries give the detailed formats of all elements of the report and the sources of all information for the report.

An RD entry in the Report Section is analogous to an FD entry in the File Section; it is the highest level of hierarchical organization for the report. The report-name specified in each RD entry must be unique.

A level 01 report group description entry is analogous to a level 01 data record description entry in the File Section. A level 01 report item is called a report group. The hierarchical definition of the report group is completed with a series of subordinate entries with levels 02-49.

An item with no subordinate items (even if its level is 01) is an elementary item. Any report item whose entry is followed by subordinate entries is a group item.

Since several reports may be defined in the Report Section, the skeletal format of the Report Section is as follows:

```
REPORT SECTION.
RD report-name-1...
01 report-group-name...
 02...
 .
 .
01...
.
RD report-name-2...
01...
.
RD report-name-n...
.
.
```

} Complete description of first report

A GENERATE statement refers to the data-name of a level 01 detail report item (a report group). For summary reporting, GENERATE may refer instead to the report-name of an RD entry instead of a detail report group data-name. The order in which level 01 report groups are specified for a given report is not significant.

Within each report group, items to be printed must be described from left to right. If the report group contains multiple lines, they must be described in order from top to bottom.

The length of each line is determined by the compiler, and depends upon the capacity of the line printers. Spaces are assumed except where a specific item is to be printed. (In a data record, on the other hand, every character position must be described.)

### RD Entries

The description of each report begins with an RD entry. Except for the level indicator (RD) and the report-name, all clauses in an RD entry are optional.

The optional clauses in an RD entry are:

| <u>Clause</u> | <u>Function</u>                                                                                                                                           |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------|
| CODE          | To assign a unique letter or digit to label each line of this report on intermediate storage. (The code character does not appear in the printed report.) |
| CONTROL(S)    | To specify data-names of control items, in the order from most significant to least significant.                                                          |
| PAGE LIMIT(S) | To specify the maximum number of lines per page.                                                                                                          |
| HEADING       | To specify the line number at which page or overflow headings may begin.                                                                                  |
| FIRST DETAIL  | To specify the line number at which detail and control lines may begin.                                                                                   |
| LAST DETAIL   | To specify the line number beyond which detail and control heading lines must not be printed.                                                             |
| FOOTING       | To specify the line number beyond which control footing lines must not be printed.                                                                        |

### Report Group Entries

Typically, the description of a report includes two or more level 01 report group entries, each followed by a hierarchy of subordinate entries. Depending upon a number of factors, most clauses (except the level-number clause) are optional. In most entries, the data-name is optional and is normally omitted. A data-name is specified in level 01 detail report group entries and in certain other entries, described later.

At the 01 report group level, the following clause is required:

| <u>Clause</u> | <u>Function</u>                                                                      |
|---------------|--------------------------------------------------------------------------------------|
| TYPE          | To specify the purpose of this report group (detail, page or control heading, etc.). |

The optional clauses in a report group entry are:

| <u>Clause</u>         | <u>Function</u>                                                                                                                                                                            |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| LINE NUMBER           | To specify vertical spacing (slewing) that is to precede production of this report group.                                                                                                  |
| NEXT GROUP            | To specify vertical spacing that is to follow production of this report group.                                                                                                             |
| COLUMN NUMBER         | To specify that this item is to be printed, and to specify its horizontal position on the line.                                                                                            |
| BLANK WHEN ZERO       | To cause this item's value to be 'spaces' when the SOURCE or SUM with which it is associated has the value zero.                                                                           |
| GROUP INDICATE        | To cause this item to be printed only at the top of the page and just after each control break.                                                                                            |
| JUSTIFIED RIGHT       | To override normal left justification when this item is edited for output.                                                                                                                 |
| PICTURE               | To specify the desired output format for this item.                                                                                                                                        |
| RESET                 | To specify control breaks where the control total is to be reset to zero.                                                                                                                  |
| SOURCE, SUM, or VALUE | To specify the source of data for this item:<br><ol style="list-style-type: none"><li>1. SOURCE - a data item.</li><li>2. SUM - a 'control total'.</li><li>3. VALUE - a literal.</li></ol> |
| USAGE                 | To specify DISPLAY-1 format when necessary.                                                                                                                                                |

## ELEMENTS OF A REPORT

### Report Groups

Each integral unit of data presented in a report, such as a page heading or footing, control heading or footing, or detail line is called a report group. A report group may consist of one or several actual lines in the printed report. In the Report Section, the first entry of each report group has level 01.

The TYPE clause is a required part of each level 01 report group description entry. The TYPE clause identifies the report group as detail or as report, page, overflow, or control heading or footing. Each report must contain at least one TYPE DETAIL report group. All other types are optional. A given heading type may be used with or without the corresponding footing, and vice versa. A report may have several distinct detail report groups or control heading or footing report groups, but no more than one of each of the other types.

### Control Data Items

Each control heading or footing is associated with a specific control data item. A control item may be any item described in the File Section or Working-Storage Section.

Control items are related to the report by a list of control data-names specified in the CONTROL(S) clause of the RD entry. When control items are specified, the reporting procedures in the object program automatically monitor all control items for changes in value.

The most significant possible control level is associated with the reserved word FINAL, which may optionally be specified in the RD entry's CONTROL(S) clause and in a control heading and/or a control footing report group description entry.

Any control item may be associated with a specific control heading and/or a specific control footing report group. (Control report groups may be specified for each control item, for none, or for any subset of control items.) Control footings may call for automatic accumulation of control totals.

Control heading report groups are presented in the following hierarchical order:

FINAL CONTROL HEADING  
MAJOR CONTROL HEADING  
.  
MINOR CONTROL HEADING

Control footing report groups are presented in the following hierarchical order:

MINOR CONTROL FOOTING  
.  
MAJOR CONTROL FOOTING  
FINAL CONTROL FOOTING

A control break is recognized whenever a control item has changed in value between execution of the previous GENERATE statement and the current GENERATE statement.

If the item producing a control break is not the least significant (rightmost) item in the list of all control items, then a control break has occurred at all less significant levels as well.

A control break causes the following automatic actions:

1. Rolling forward of control totals.
2. Presentation of control footings up through the control break level.
3. Resetting control totals to zero, up through the control break level.
4. Presentation of control headings from the control break level down through the least significant control level.

When it is specified, a final control heading is presented only once for the report, upon the first execution of a GENERATE statement. Similarly, a final control footing is presented only once, upon execution of the TERMINATE statement.

When the TERMINATE statement is executed, a final control break is understood to have occurred, so all control footing report groups are then presented, in order from least significant through final.

When a control break occurs, control footings must use the old values of the control data items, while control headings use the new values of the control data items. A special provision causes old control item values to be retained for control footings. No such provision exists for items which are not control items.

#### Page Breaks and Overflow Breaks

If page heading and/or page footing report groups are specified, they are automatically presented at the top or bottom of each page of the printed report.

A 'control group' consists of all control heading, detail, and control footing report groups produced between two successive control breaks (including the control footing(s) produced by the latter).

The bottom of a page may be reached either between control groups or within a control group. In some cases, one set of page heading and/or footing formats can be specified if a page break occurs between control groups, and a different set of headings and/or footings can be specified if a page break occurs within a control group. Headings and footings in the latter category are called 'overflow' headings and footings. If overflow headings and footings have been specified, a page break within a control group is considered to have produced an 'overflow' condition.

On a given page, either a page heading or an overflow heading may appear, but not both. Similarly, either a page footing or an overflow footing may appear, but not both. (If overflow headings are not specified but page headings are specified, each page break produces a page heading. The equivalent rule applies for overflow and page footings.)

If overflow headings or footings are specified for a report, the RD entry must include the LAST DETAIL phrase of the PAGE LIMITS clause.

If a report has both page and overflow headings and/or both page and overflow footings, the FOOTING phrase of the PAGE LIMITS clause affects detection and results in an overflow condition as follows:

1. If the FOOTING phrase is omitted, an overflow condition exists whenever the current detail or control footing report group cannot fit on the current page.
2. If the FOOTING phrase is specified, an overflow condition exists if the current report group is a detail which cannot fit within the LAST DETAIL phrase limit. If the current report group is a control footing, an overflow condition exists if the entire set of control footing report groups produced by this control break cannot fit on the current page (within the FOOTING phrase limit). This feature may be used to force all control footings to appear within a single page.

When an overflow condition occurs, it exists from the presentation of the last element of the current control group on one page to the presentation of the first element of the same control group on the next page.

Except when an overflow condition is determined as described above, a page break causes presentation of page footings and headings.

### File Characteristics

Each report is produced on an output file by the object program. The output file must be described by at least an FD entry in the File Section of the Data Division plus associated Environment Division paragraphs and phrases.

A given output file may receive one or more reports. The REPORT(S) clause of the FD entry lists the report or reports belonging to the file. This is the only explicit relationship between a report and the file to which it belongs. (Although a GENERATE statement does not mention the output file, any necessary 'writes' to the file are implied by each execution of a GENERATE statement.)

For a file receiving multiple reports, it is necessary to label each report line uniquely so that the lines belonging to the respective reports can be distinguished for printing. The CODE clause of the RD entry is used for that purpose. With the CODE clause, the user can associate a unique letter or numeric digit with each report; the compiler then causes every line of the report to be labeled in a standard manner with the unique character assigned by the user. The code character appears in intermediate storage only (e.g., SYSOUT or magnetic tape), not in the printed report.

All files referenced by Report Writer clauses or statements must have a process area applied, either by explicitly specifying the APPLY PROCESS AREA phrase or implicitly by specifying the FOR CARDS or FOR LISTING phrase.

### Line Counter

A line counter is implicitly provided for each report. It is used by the generated reporting procedures to recognize page (and overflow) breaks, and to control vertical page format.



The line counter is automatically set to zero initially, and it is reset to zero whenever a page break occurs. It is automatically set, reset, and incremented on the basis of values specified in the LINE NUMBER and NEXT GROUP clauses in the respective report groups. It is automatically tested on the basis of values specified in the PAGE LIMITS clause of the RD entry.

A page break occurs whenever a relative LINE NUMBER or relative NEXT GROUP value causes the line counter to exceed a relevant PAGE LIMITS value.

The fixed data-name LINE-COUNTER may be referred to if it is necessary to access the line counter contents. The report-name may be used as a qualifier for LINE-COUNTER; such qualification is necessary whenever the Report Section includes more than one report.

If the last line produced had no relevant NEXT GROUP clause, the line counter value is the number of the last line printed. Otherwise, the line counter value is the number of the last line skipped.

Procedure Division statements should never change the value of a line counter. Otherwise, an unpredictable loss of page format control may occur.

### Page Counter

A page counter is implicitly provided for each report. It is primarily used as a SOURCE data item within page heading report groups, to provide consecutive page numbers for the report.

The initial value of the page counter is one. Its value is automatically incremented by one each time a page break occurs. (The increment follows production of any page or overflow footing, but precedes production of any page or overflow heading.)

The fixed data-name PAGE-COUNTER is referred to in a SOURCE clause or in the Procedure Division to access the page counter value. The report-name may be used as a qualifier for PAGE-COUNTER; such qualification is necessary whenever the Report Section includes more than one report.

Normally, Procedure Division statements should not change the value of a page counter. However, a Procedure Division statement may change the starting value of a page counter if an initial page number other than one (1) is desired.

## REPORT WRITER EFFICIENCY TECHNIQUES

### SUM Counter Manipulation

A function of the Report Writer that must be clarified to avoid producing inefficient object code is the manipulation of SUM counters. There are three distinct types of SUM counter manipulation; subtotalling, rolling forward, and crossfooting. A definition and illustration of each type of manipulation is presented below.

## SUBTOTALLING

Subtotalling is the most basic type of SUM counter manipulation. In this method, a SUM counter is augmented by the value of the SUM operand for each execution of a GENERATE statement of the TYPE DETAIL report group which contains the SOURCE counterpart of the SUM operand.

### Example:

```
01 DETAIL-1 TYPE DETAIL LINE PLUS 1.
 02 SOURCE IS COST.
 .
01 MINOR TYPE CF MINR LINE PLUS 1.
 02 SCTR-1 COLUMN 50 PIC Z(6).99 SUM COST.
 .
01 INTERMEDIATE TYPE CF INTRM LINE PLUS 1.
 02 SCTR-2 COLUMN 50 PIC Z(6).99 SUM COST.
 .
01 MAJOR TYPE CF MAJR LINE PLUS 1.
 02 SCTR-3 COLUMN 50 PIC Z(6).99 SUM COST.
 .
01 FIN-TOT TYPE CF FINAL LINE PLUS 1 NEXT GROUP NEXT PAGE.
 02 SCTR-4 COLUMN 50 PIC Z(6).99 SUM COST.
 .
 .
```

At each execution of a GENERATE DETAIL-1, the value of COST will be added into SUM counters SCTR-1, SCTR-2, SCTR-3, and SCTR-4. When a control break occurs, no 'rolling forward' of counters is necessary since all counters are effectively 'subtotalled'. The only remaining actions to be performed are:

1. Presentation of the control footing report groups from the least inclusive (MINOR) up through the control footing representing the control break level.
2. Resetting the corresponding SUM counters to zero after each control footing is presented.

## ROLLING FORWARD

Rolling forward is a type of SUM counter manipulation in which SUM counters defined in control footing report groups of lower control levels are added to SUM counters defined in control footing report groups of higher control levels during control break processing.

In the previous example, for instance, the identical results may be obtained more efficiently by 'rolling forward' the SUM counters.

Example:

```
01 DETAIL-1 TYPE DETAIL LINE PLUS 1.
02 SOURCE IS COST.
.
01 MINOR TYPE CF MINR LINE PLUS 1.
02 SCTR-1 COLUMN 50 PIC Z(6).99 SUM COST.
.
01 INTERMEDIATE TYPE CF INTRM LINE PLUS 1.
02 SCTR-2 COLUMN 50 PIC Z(6).99 SUM SCTR-1.
.
01 MAJOR TYPE CF MAJR LINE PLUS 1.
02 SCTR-3 COLUMN 50 PIC Z(6).99 SUM SCTR-2.
.
01 FIN-TOT TYPE CF FINAL LINE PLUS 1 NEXT GROUP NEXT PAGE.
02 SCTR-4 COLUMN 50 PIC Z(6).99 SUM SCTR-3.
.
.
```

The following sequence of events occurs in the above example:

1. At each execution of a GENERATE DETAIL-1 statement, the value of COST is added into SUM counter SCTR-1 (subtotalling).
2. When a control break occurs on control data-name MINR, the control footing report group called MINOR is presented; then SUM counter SCTR-1 is added (rolled forward) to SUM counter SCTR-2.
3. When a control break occurs at a higher control break level, the control footing report groups are presented in sequence from the inclusive (MINOR) up to and including the control footing at which the control break occurred. After each control footing is presented, the SUM counters for that report group are rolled forward to corresponding SUM counters in higher level control footing report groups.

Thus, the subtotalling operation occurs only at the least inclusive (MINOR) control break level. The remaining SUM counters are augmented only when control break processing takes place.

## CROSSFOOTING

Crossfooting is a type of SUM counter manipulation in which SUM counters defined in a given control footing report group are added to other SUM counters in the same report group during control break processing.

Example:

```
01 DETAIL-1 TYPE DETAIL LINE PLUS 1.
 02 SOURCE IS COST-1.
 02 SOURCE IS COST-2.
 .
01 MINOR TYPE CF MINR LINE PLUS 1.
 02 SCTR-1 COLUMN 50 PIC Z(6).99 SUM COST-1.
 02 SCTR-2 COLUMN 60 PIC Z(6).99 SUM COST-2.
 02 SCTR-3 COLUMN 70 PIC Z(9).99 SUM SCTR-1, SCTR-2.
 .
01 INTERMEDIATE TYPE CF INTRM LINE PLUS 1.
 02 SCTR-4 COLUMN 50 PIC Z(6).99 SUM SCTR-1.
 02 SCTR-5 COLUMN 60 PIC Z(6).99 SUM SCTR-2.
 02 SCTR-6 COLUMN 70 PIC Z(9).99 SUM SCTR-4, SCTR-5.
 .
 .
```

The following sequence of events occurs in the above example:

1. At each execution of a GENERATE DETAIL-1 statement, SUM counters SCTR-1 and SCTR-2 are augmented by the corresponding values of COST-1 and COST-2 (subtotalling).
2. When a control break occurs for the control footing report group called MINOR, SUM counters SCTR-1 and SCTR-2 are added into SUM counter SCTR-3 before the report group is presented (crossfooting).
3. After the report group called MINOR is presented, SUM counters SCTR-1 and SCTR-2 are added into SUM counters SCTR-4 and SCTR-5, respectively (rolled forward).
4. SUM counters SCTR-1, SCTR-2, and SCTR-3 are reset to zero.
5. When a control break occurs for the control footing report group called INTERMEDIATE, SUM counters SCTR-4 and SCTR-5 are added into SUM counter SCTR-6 before the report group is presented (crossfooting).

#### SOURCE/SUM Correlation

A common source of error in Report Writer programs results from a misunderstanding of the SOURCE/SUM correlation concept. This concept, simply stated, requires that a SUM operand must either be:

1. The object of a SOURCE IS clause in a TYPE DETAIL report group, or
2. The name of a SUM counter defined in a lower level control footing report group.

The source error which occurs most frequently is that a given report description (RD) contains more than one TYPE DETAIL report group, and a given SUM operand appears as the object of a SOURCE IS clause in more than one TYPE DETAIL report group.

Example:

```
RD REPORT-1 CONTROLS ARE MINR...
01 DETAIL-1 TYPE DE LINE PLUS 1.
 02 SOURCE IS COST.
 .
 .
01 DETAIL-2 TYPE DE LINE PLUS 1.
 02 SOURCE IS COST.
 .
 .
01 MINOR TYPE CF MINR LINE PLUS 1.
 02 SCTR-1 COLUMN 50 PIC Z(6).99 SUM COST.
 .
 .
```

For each execution of either a GENERATE DETAIL-1 or a GENERATE DETAIL-2 statement, the SUM counter SCTR-1 will be augmented by the value of COST since it is the object of a SOURCE IS clause in both TYPE DETAIL report groups.

The UPON phrase of the SUM clause may be used to selectively augment a given SUM counter.

If the previous (last) example is changed to read:

```
01 MINOR TYPE CF MINR LINE PLUS 1.
 02 SCTR-1 COLUMN 50 PIC Z(6).99 SUM COST UPON DETAIL-1.
 .
 .
```

this definition indicates that SUM counter SCTR-1 will be augmented only when a GENERATE statement is executed for DETAIL-1.

### Pre-Slew and Post-Slew Algorithms

The algorithms used by the COBOL Report Writer in making pre- and post-slew calculations are presented below, with related examples.

#### PRE-SLEW CALCULATIONS

The basic algorithm is:

LINE PLUS N slews N-1 lines.

Therefore:

LINE PLUS 0 → pre-slew 0 lines.  
LINE PLUS 1 → pre-slew 0 lines.  
LINE PLUS 2 → pre-slew 1 line.  
LINE PLUS 3 → pre-slew 2 lines.

Exception:

LINE PLUS 0 → pre-slew 0 lines NOT N-1 lines.

#### POST-SLEW CALCULATIONS

The basic algorithm is:

NEXT GROUP PLUS M slews M lines.

Therefore:

NEXT GROUP PLUS 0 → post-slew 0 lines.  
NEXT GROUP PLUS 1 → post-slew 1 line.  
NEXT GROUP PLUS 2 → post-slew 2 lines.  
NEXT GROUP PLUS 3 → post-slew 3 lines.

Exception:

If NEXT GROUP not specified → automatic post-slew 1 line.

#### COMBINATIONS OF PRE-SLEW AND POST-SLEW CALCULATIONS

The combinations of pre- and post-slew line calculations are presented below:

LINE PLUS 0 → pre-slew 0, post-slew 1  
LINE PLUS 1 → pre-slew 0, post-slew 1  
LINE PLUS 2 → pre-slew 1, post-slew 1  
LINE PLUS 0  
    NEXT GROUP PLUS 0 → pre-slew 0, post-slew 0  
LINE PLUS 1  
    NEXT GROUP PLUS 0 → pre-slew 0, post-slew 0  
LINE PLUS 2  
    NEXT GROUP PLUS 0 → pre-slew 1, post-slew 0  
LINE PLUS 1  
    NEXT GROUP PLUS 1 → pre-slew 0, post-slew 1  
LINE PLUS 2  
    NEXT GROUP PLUS 1 → pre-slew 1, post-slew 1  
LINE PLUS 1  
    NEXT GROUP PLUS 2 → pre-slew 0, post-slew 2  
LINE PLUS 2  
    NEXT GROUP PLUS 2 → pre-slew 1, post-slew 2

Thus:

1. For normal single spacing of a report, a LINE PLUS 1 designation is most often used. It actually gives a pre-slew of 0; however, the implicit post-slew of 1 yields the desired result.
2. For normal double spacing of a report, a LINE PLUS 2 gives a pre-slew of 1 which, when added to implicit post-slew of 1 from the previous line, results in a double-spaced report.
3. If line overprint is desired, the detail line must be split into two TYPE DETAIL report groups. This may be accomplished by using a LINE PLUS 0 NEXT GROUP PLUS 0 on the first TYPE DETAIL to be generated. The remainder of the detail line generated by the second TYPE DETAIL would specify LINE PLUS 0. This would result in both TYPE DETAIL lines being printed on the same line.

#### REPORT WRITER TABLE CONSTRAINTS

Several fixed-length table restrictions apply due to the size and complexity of the Report Writer system. They are given below.

#### SUM Operand Limitations

1. In a given report, the same operand must not appear in over ten SUM clauses.

Example:

```
01 DETAIL-1 TYPE DE LINE PLUS 1.
02 SOURCE IS COST.
:
.
```

If the identifier COST were to be used as the object of a SUM clause in over ten separate statements, the following error message is printed:

```
ER - REFERENCES TO SUM OPERAND EXCEED LIMITS
NO SUMMING WILL TAKE PLACE FOR THIS SUM CLAUSE.
```

This limit may be reached in cases where subtotalling is being used in many levels of control footings. Normally, the problem can be resolved by rolling forward the SUM counters.

2. In a given report, no more than 100 SOURCE/SUM correlations may be specified. Specification of a SUM clause whose operand correlates (matches) with the same operand that appears in more than one SOURCE clause will result in a separate entry being placed in the SUM stack for each SOURCE/SUM correlation. Similarly, SUM clauses that specify multiple operands will result in a separate entry being placed into the SUM stack for each SOURCE/SUM correlation. If a program exceeds this limit, the following error message is printed:

ER - NUMBER OF SOURCE/SUM CORRELATIONS IN THIS REPORT  
EXCEEDS LIMIT - NO SUMMING OCCURS FOR THIS STATEMENT.

If this limitation is reached, the user may perform required calculations in a USE BEFORE REPORTING statement for those SUM counters over the limit. The user-defined (working-storage) SUM counter may be presented at control break time by using it as the object of a SOURCE clause in the control footing.

### RESET Stack Limitations

The RESET clause, which is used in conjunction with the SUM clause, is processed by an internal stack mechanism within the Report Writer. The RESET stack handles a maximum of ten RESET clauses within a given report and has no overflow capability. If the capacity of the RESET stack is exceeded, the error message

\*\*\*\*\* ER NUMBER OF RESETS EXCEEDS STACK CAPACITY  
RESET CLAUSE IS IGNORED.

is printed for each RESET clause encountered after the limit of ten has been reached.

If this limitation is experienced, the user may circumvent the problem at the source program level by using the following procedures:

1. Define working-storage SUM counters for all SUM counters that exceed the RESET stack capacity.
2. Augment the working-storage SUM counters by Procedure Division statements that are outside Report Writer control.
3. Refer to the working-storage SUM counters by specifying the SOURCE clause in the control footing in which it is to be presented.
4. Define a USE BEFORE REPORTING declarative section for the control break level at which resetting is desired.
5. Within the USE BEFORE REPORTING statement, move zeros to the working-storage SUM counters that require manual resetting.



Example:

```
REPORT SECTION.
.
.
01 MINOR-CONTROL TYPE CF MINOR...
02 LINE PLUS 1.
03 A COLUMN 1 PIC Z(6).99 SUM COST RESET ON INTERMEDIATE.
.
.
01 INTERMEDIATE-CONTROL TYPE CF INTERMEDIATE...
02 LINE PLUS 1.
03 B COLUMN 1 PIC Z(6).99 SUM COST RESET ON FINAL.
.
.
01 FINAL-CONTROL TYPE CF FINAL...
02 LINE PLUS 1.
.
.
.
```

In case of RESET stack overflow, the automatic resetting for SUM counters A and B may be accomplished manually by incorporating the following changes in the source program:

```
WORKING-STORAGE SECTION.
77 A PIC 9(6)V99 VALUE 0.
77 B PIC 9(6)V99 VALUE 0.

REPORT SECTION.
.
.
01 MINOR-CONTROL TYPE CF MINOR...
02 LINE PLUS 1.
03 COLUMN 1 PIC Z(6).99 SOURCE IS A.
.
.
01 INTERMEDIATE-CONTROL TYPE CF INTERMEDIATE...
02 LINE PLUS 1.
03 COLUMN 1 PIC Z(6).99 SOURCE IS B.
.
.
01 FINAL-CONTROL TYPE CF FINAL...
.
.
.

PROCEDURE DIVISION.
DECLARATIVES.
INT-CTL SECTION.
USE BEFORE REPORTING INTERMEDIATE-CONTROL.
PARA-1. MOVE ZEROS TO A.
FIN-CTL SECTION.
USE BEFORE REPORTING FINAL-CONTROL.
PARA-2. MOVE ZEROS TO B.
END DECLARATIVES.
.
.
BEGIN. READ INPUT-FILE.
COMPUTE A = A + COST.
COMPUTE B = B + COST.
GENERATE...
.
.
```

There is one significant difference in the manual reset method compared to the automatic reset method. Automatic resetting occurs after the control footing print line has been presented. Manual resetting, at USE BEFORE REPORTING time, occurs before the control footing print line is presented.

### Report Table Capacity

The Report Table is a fixed-length memory area within the compiler in which all Report Writer Data Division entries are constructed. The table has a total capacity of 1900 words, consisting of a 640-word fixed entry portion and a 1260-word variable entry portion. The Report Table is designed so that data in the fixed portion is inserted from the top to the bottom of the table and the data in the variable portion is inserted from the bottom to the top. The Report Table has no overflow capability.

When the RD level indicator is encountered in the source program, a ten-word entry is placed into the fixed portion of the Report Table as the report-name entry. If the CONTROL(S) clause is specified, a nine-word control entry is placed into the fixed portion of the table, followed by an additional 13-word entry for each control data-name specified. When a report group level indicator is encountered, a ten-word entry for that report group is placed into the fixed portion. A corresponding entry for the report group description entry is placed into the variable portion of the table.

Each individual report group entry must be small enough to be contained within the variable portion of the table. When a given report group is built in its entirety, the data in the variable portion is written to an internal intermediate Report Writer file and the next report group encountered is inserted into the variable portion. The fixed portion must contain entries for all report groups defined within a given RD and, consequently, is not written to the intermediate file until a new RD or the Procedure Division header is encountered. The variable portion of the Report Table is always at least 1260 words in length. However, the variable portion is automatically enabled to utilize the unused portion of the fixed portion, up to a maximum size of 1650 words, if additional memory space is required to contain a very large report group. Since the length of the fixed portion increases with each report group encountered, it is recommended that the largest report group descriptions be described near the beginning of the report description entry.

If the combined capacity of the variable portion and fixed portion is exhausted during the processing of a given report group, or if the variable report group size exceeds 1650 words, the following error message is printed:

```
***** REPORT GROUP DESCRIPTION EXCEEDS COMPILER CAPACITY - DELETED
 REMAINING ENTRIES UNDER CURRENT 01.
```

When this error condition occurs, it is necessary to reduce the size of the report group as described in the Calculation of Report Group Size paragraph in this section.

The capacity of the fixed portion of the Report Table may also be exceeded in cases where a report description contains a large number of report groups. For example, a report having no CONTROL(S) clause can contain a maximum of 62 report groups. This capacity is reduced if control data-names are specified. If the fixed portion (640 words maximum) is exceeded, the following error message is printed:

```
***** OVERALL REPORT DESCRIPTION EXCEEDS COMPILER CAPACITY - DELETED
 REMAINING ENTRIES UNDER CURRENT RD.
```

When this error condition occurs, it is necessary to reduce the number of specified report groups either by combining the groups or by dividing the RD into two separate reports.

Since large report group description entries, normally TYPE DETAIL or control footing, occasionally overflow the Report Table capacity, the following discussion is intended to assist the user in structuring report group descriptions to fit within the capacity of the Report Table.

#### REPORT GROUP ENTRIES

A report group entry is built for the 01 entry which contains the TYPE clause. Only one such entry appears per report group. The entry varies from a minimum of eight words to a maximum of 11 words.

A basic 01 report group entry containing only a TYPE statement is built as an eight-word report group entry.

Example:

```
01 DET-L TYPE DE.
```

If a LINE or NEXT GROUP designation or a combination of both appear in the 01 report group statement, two additional words are required in the entry being built.

Example:

```
01 DET-X TYPE DE LINE PLUS 1. (or)
01 DET-Y TYPE DE NEXT GROUP PLUS 2. (or)
01 DET-Z TYPE DE LINE PLUS 1 NEXT GROUP PLUS 2.
```

All of these statements build a ten-word entry.

If the report group is a control footing report group, one additional word is required in the entry being built, which is used as a total sum word count.

Example:

```
01 TYPE CF data-name LINE PLUS 1 NEXT GROUP NEXT PAGE.
```

This statement is built as an 11-word entry.

#### GROUP ENTRIES

When structuring a report group, the user may specify intervening group levels. Each group entry at the 02 level or below requires a group entry that is nearly identical to the report group entry described above. It may vary in size from eight words minimum to ten words maximum. For example, consider the following structure:

```
01 CTL-X TYPE CF CNTRL NEXT GROUP NEXT PAGE.
 02 LINE PLUS 1.
 03 COLUMN 1 PICTURE Z(6) SUM data-name.
02 LINE PLUS 2.
 03 ...
```

The 01 TYPE entry would build an entry 11 words in length since it has a NEXT GROUP clause and is a TYPE control footing. The 02 group statement builds a ten-word entry inheriting most of its data from the report group entry and adding the word of data to describe the LINE clause.

#### SOURCE ENTRIES

A source entry is built for each elementary entry containing a SOURCE IS clause. These entries can vary greatly in size, from a minimum entry of 27 words to a maximum entry of 74 words.

The most basic SOURCE clause is found in a detail report group. It does not contain a COLUMN clause and is therefore not printed. A SUM counter must be defined for a control footing report group.

Example:

```
02 SOURCE IS data-name.
```

This statement is built as a 27-word entry.

When the SOURCE clause contains subscripted items, the number of words required increases rapidly. One data-name subscript adds 12 additional words to the entry built.

Example:

```
02 SOURCE IS data-name (data-name). 39 words
```

A single literal subscript requires nine additional words.

Example:

```
02 SOURCE IS data-name (literal). 36 words
```

Each subsequent subscript in the same entry adds 11 words in the case of a data-name or eight words in the case of a literal.

Example:

```
02 SOURCE IS data-name (dn,dn-1). 50 words
02 SOURCE IS data-name (lit,lit-1). 44 words
02 SOURCE IS data-name (dn,lit). 47 words
02 SOURCE IS data-name (dn,dn-1,dn-2). 61 words
02 SOURCE IS data-name (lit,lit-1,lit-2). 52 words
02 SOURCE IS data-name (dn,dn-1,lit). 58 words
02 SOURCE IS data-name (dn,lit,lit-1). 55 words
```

When a COLUMN clause is added, it designates that a receiving field must be provided in the source entry being built. The entry varies from ten to 13 words in length, depending on whether or not editing is required in the receiving field.

Example:

```
02 COLUMN 1 PICTURE 9(6) SOURCE IS data-name.
```

This entry would not require editing, so the total entry built would be 27 words plus ten or 37 words. If the statement were

```
02 COLUMN 1 PICTURE ZZZ,ZZ9.99 SOURCE IS data-name.
```

editing is required; thus, the entry would require 27 words plus 13 or 40 words.

The largest possible entry would therefore be of the type:

```
02 COLUMN 1 PICTURE ZZ,999 SOURCE IS dn (dn,dn-1,dn-2).
```

This would require 74 words.

#### SUM ENTRIES

A sum entry is built for each elementary entry containing a SUM clause. Sum entries can vary in size from a minimum of 37 words to a maximum of 40 words.

The basic SUM clause is found in the TYPE control footing report group and is usually of the format:

```
02 CTL-X COLUMN 1 PICTURE ZZZ,ZZ9.99 SUM data-name.
```

or

```
02 COLUMN 1 PICTURE ZZZ,ZZ9.99 SUM data-name.
```

Either of the above statements will be built as an entry 40 words in length.

The addition of subscripts, either data-name or literal, does not increase the size of the entry to be built. Thus, the statement

```
02 COLUMN 1 PICTURE Z(6).99 SUM dn (dn,dn-1,dn-2).
```

requires the same number of words as

```
02 COLUMN 1 PICTURE Z(6).99 SUM data-name.
```

The variance between 37- and 40-word entries is due to receiving field editing requirements and is described above in the Source Entries paragraph.

## VALUE ENTRIES

A VALUE entry is built for each elementary item that contains a VALUE clause. Since the VALUE clause expresses a literal that can range from a single character to 132 characters in length, it follows that the entry built for a VALUE clause varies in the same proportion. The example

```
02 COLUMN 1 PICTURE X VALUE "-".
```

represents a minimum entry and requires 26 words. On the other hand, the statement

```
02 COLUMN 1 SIZE 132 VALUE "132 character literal---".
```

represents a maximum entry and requires a 48-word entry.

The entry sizes for the various literal sizes are listed below:

| <u>Literal Size<br/>(Characters)</u> | <u>VALUE Entry<br/>(Words)</u> |
|--------------------------------------|--------------------------------|
| 1-2                                  | 26                             |
| 3-8                                  | 27                             |
| 9-14                                 | 28                             |
| 15-20                                | 29                             |
| 21-26                                | 30                             |
| 27-32                                | 31                             |
| 33-38                                | 32                             |
| 39-44                                | 33                             |
| 45-50                                | 34                             |
| 51-56                                | 35                             |
| 57-62                                | 36                             |
| 63-68                                | 37                             |
| 69-74                                | 38                             |
| 75-80                                | 39                             |
| 81-86                                | 40                             |
| 87-92                                | 41                             |
| 93-98                                | 42                             |
| 99-104                               | 43                             |
| 105-110                              | 44                             |
| 111-116                              | 45                             |
| 117-122                              | 46                             |
| 123-128                              | 47                             |
| 129-132                              | 48                             |

## EXCEPTIONS TO ENTRY SIZES

Two possible exceptions may change the maximum entry sizes as indicated in this discussion. Both are rather remote in usage but should be clarified:

1. If an elementary SOURCE, SUM, or VALUE entry contains a LINE clause as part of its description, the entry built is increased by two words.

Example:

```
02 LINE PLUS 1 COLUMN 1 PICTURE X(6) SOURCE dn.
```

This entry builds a source entry of 39 words as compared to 37 words if the LINE clause does not appear at the elementary level.

SUM and VALUE entries reflect the same two-word increase in size.

2. If editing is required in the receiving field description designated by the COLUMN and PICTURE clauses, and the receiving field is over 38 characters in length, one additional word is required. If the receiving field is over 76 characters in length, two additional words are required.

## CALCULATION OF REPORT GROUP SIZE

The length of the variable entry portion of the Report Table in most cases is 1650 words; however, in certain instances, it can be reduced to a minimum of 1260 words. This reduction occurs only when the following conditions exist:

1. The report description (RD) contains a large number of report groups.
2. The larger report groups appear near or at the end of the report description.

A rule to follow in report description organization is to place the larger report groups at the beginning of the report description. In this way, the maximum variable Report Table size will nearly always be available.



The following example shows the calculation of report group size with a table capacity of 1260 words:

```

01 DETL-X TYPE DE. 8
02 LINE PLUS 2. 10
03 COLUMN 1 PICTURE Z(6).99 SOURCE dn (dn,dn-1,dn-2). 74
03 COLUMN 15 PICTURE Z(6).99 SOURCE dn-1 (dn,dn-1,dn-2). 74
03 COLUMN 30 PICTURE Z(6).99 SOURCE dn-2 (dn,dn-1,dn-2). 74
03 COLUMN 45 PICTURE Z(6).99 SOURCE dn-3 (dn,dn-1,dn-2). 74
03 COLUMN 60 PICTURE Z(6).99 SOURCE dn-4 (dn,dn-1,dn-2). 74

02 LINE PLUS 3. 10
03 COLUMN 1 PICTURE Z(6).99 SOURCE dn-5 (dn,dn-1,dn-2). 74
03 COLUMN 15 PICTURE Z(6).99 SOURCE dn-6 (dn,dn-1,dn-2). 74
03 COLUMN 30 PICTURE Z(6).99 SOURCE dn-7 (dn,dn-1,dn-2). 74
03 COLUMN 45 PICTURE Z(6).99 SOURCE dn-8 (dn,dn-1,dn-2). 74
03 COLUMN 60 PICTURE Z(6).99 SOURCE dn-9 (dn,dn-1,dn-2). 74

02 LINE PLUS 4. 10
03 COLUMN 1 PICTURE Z(6).99 SOURCE dn-10 (dn,dn-1,dn-2). 74
03 COLUMN 15 PICTURE Z(6).99 SOURCE dn-11 (dn,dn-1,dn-2). 74
03 COLUMN 30 PICTURE Z(6).99 SOURCE dn-12 (dn,dn-1,dn-2). 74
03 COLUMN 45 PICTURE Z(6).99 SOURCE dn-13 (dn,dn-1,dn-2). 74
03 COLUMN 60 PICTURE Z(6).99 SOURCE dn-14 (dn,dn-1,dn-2). 74

02 LINE PLUS 5. 10
03 COLUMN 1 PICTURE Z(6).99 SOURCE dn-15 (dn,dn-1,dn-2). 74
→ 03 COLUMN 15 PICTURE Z(6).99 SOURCE dn-16 (dn,dn-1,dn-2). 74
03 COLUMN 30 PICTURE Z(6).99 SOURCE dn-17 (dn,dn-1,dn-2). 74

```

At this point, report group capacity is exceeded and the detail report group must be subdivided into two detail report groups. Subdivision can be accomplished by dividing the report group into two report groups of the same type. In the case of the TYPE DETAIL report group, it requires insertion of an "01 data-name TYPE DE." statement and an additional GENERATE statement in the Procedure Division. If the overflowed group is control footing, the subdivision is more complex. A dummy CONTROL data-name with the same PICTURE and USAGE as the original must be defined in working-storage. Immediately after each READ of the pertinent input file, the field which makes up the original CONTROL data-name must be moved to the dummy CONTROL data-name in working-storage. The dummy CONTROL data-name becomes the CONTROL data-name for the new 01 TYPE CF report group and is also inserted in the CONTROLS ARE clause of the corresponding RD entry.

For example, if the following TYPE CF report group is subdivided,

```

RD REPORT-X CONTROLS ARE FINAL, DN1, DN2.
01 TYPE CF DN1.
02 ...

```

a dummy CONTROL data-name with the same PICTURE and USAGE as the original must be defined in working-storage:

```

77 DN1A PICTURE 9(6).

```

After each READ of the pertinent input file containing CONTROL data-name DN1, the new value of DN1 must be moved to DN1A before the corresponding GENERATE statement for the report being produced:

```
READ INPUT-FILE AT END GO TO ---.
MOVE DN1 TO DN1A.
GENERATE DETAIL-1.
```

The subdivided report group would be:

```
RD REPORT-X CONTROLS ARE FINAL, DN1A, DN1, DN2.
01 TYPE CF DN1.
02 ...
.
02 ...
01 TYPE CF DN1A.
02 ...
```

The control break for DN1 and DN1A will occur at the same time. The control footing report groups are presented from minor to major; therefore, the report group with DN1 will be produced before the report group with DN1A. The order may be adjusted as necessary for program requirements.

#### REPORT WRITER PROGRAM EXAMPLE

The following sample program (RPTW68) is a guide for explaining the Report Writer feature in COBOL. The program produces one report utilizing three levels of control breaks, and illustrates the interaction of the various report groups that make up the report description. In addition, if the four-page report produced by this program is examined, the effect of the vertical and horizontal line formatting features becomes evident.

COBOL  
ALT #

## SOURCE LISTING

REF COMPILER COMMENTS  
LINE #

|       |                                                           |       |                                      |
|-------|-----------------------------------------------------------|-------|--------------------------------------|
| 00001 | IDENTIFICATION DIVISION.                                  | 00001 |                                      |
| 00002 | PROGRAM-ID. RPTW68.                                       | 00002 |                                      |
| 00003 | REMARKS.                                                  | 00003 |                                      |
| 00004 | EXAMPLE COBOL REPORT WRITER SOURCE PROGRAM.               | 00004 |                                      |
| 00005 | ENVIRONMENT DIVISION.                                     | 00005 |                                      |
| 00006 | OBJECT-COMPUTER. 6000 WITH EIS.                           | 00006 |                                      |
| 00007 | SPECIAL-NAMES. "Q" IS KODE.                               | 00007 |                                      |
| 00008 | INPUT-OUTPUT SECTION.                                     | 00008 |                                      |
| 00009 | FILE-CONTROL.                                             | 00009 |                                      |
| 00010 | SELECT CARD-FILE ASSIGN TO BA FOR CARDS .                 | 00010 |                                      |
| 00011 | SELECT REPORT-FILE ASSIGN TO AB FOR LISTING.              | 00011 |                                      |
| 00012 | I-O-CONTROL.                                              | 00012 |                                      |
| 00013 | APPLY SYSTEM STANDARD ON CARD-FILE REPORT-FILE .          | 00013 |                                      |
| 00014 |                                                           | 00014 |                                      |
| 00015 | DATA DIVISION.                                            | 00015 |                                      |
| 00016 | FILE SECTION.                                             | 00016 |                                      |
| 00017 | FD CARD-FILE                                              | 00017 |                                      |
| 00018 | LABEL RECORDS ARE STANDARD, DATA RECORD IS INPUT-RECORD . | 00018 |                                      |
| 00019 | 01 INPUT-RECORD.                                          | 00019 | CONTAINS 80 CHARACTERS 14 WORDS      |
| 00020 | 02 FILLER PIC AA.                                         | 00020 | STARTS IN CHARACTER POSITION 1       |
| 00021 | 02 DEPT PIC XXX.                                          | 00021 | STARTS IN CHARACTER POSITION 3       |
|       |                                                           |       | REF BY 00102                         |
| 00022 | 02 FILLER PIC AA.                                         | 00022 | STARTS IN CHARACTER POSITION 6       |
| 00023 | 02 NO-PURCHASES PIC 99.                                   | 00023 | STARTS IN CHARACTER POSITION 8       |
|       |                                                           |       | REF BY 00103 00113                   |
| 00024 | 02 FILLER PIC A.                                          | 00024 | STARTS IN CHARACTER POSITION 10      |
| 00025 | 02 TYPE-PURCHASE PIC 9.                                   | 00025 | STARTS IN CHARACTER POSITION 11      |
|       |                                                           |       | REF BY 00104 00157                   |
| 00026 | 02 MONTH PIC 99.                                          | 00026 | STARTS IN CHARACTER POSITION 12      |
|       |                                                           |       | REF BY 00095 00099 00122 00144 00146 |
|       |                                                           |       | 00156 00158                          |
| 00027 | 02 DAY PIC 99.                                            | 00027 | STARTS IN CHARACTER POSITION 14      |
|       |                                                           |       | REF BY 00101 00112                   |
| 00028 | 02 FILLER PIC A.                                          | 00028 | STARTS IN CHARACTER POSITION 16      |
| 00029 | 02 COST PIC S999V99.                                      | 00029 | STARTS IN CHARACTER POSITION 17      |
|       |                                                           |       | REF BY 00105 00114                   |
| 00030 | 02 COST1 REDEFINES COST PIC S999V99.                      | 00030 | STARTS IN CHARACTER POSITION 17      |
|       |                                                           |       | REF BY 00116 00127 00133             |
| 00031 | 02 FILLER PIC X(59).                                      | 00031 | STARTS IN CHARACTER POSITION 22      |
| 00032 | FD REPORT-FILE                                            | 00032 |                                      |
| 00033 | LABEL RECORDS ARE STANDARD, REPORT IS EXPENSE-REPORT .    | 00033 |                                      |

COBOL  
ALT #

## SOURCE LISTING

REF COMPILER COMMENTS  
LINE #

|       |                                                     |       |                              |                         |          |
|-------|-----------------------------------------------------|-------|------------------------------|-------------------------|----------|
| 00034 | WORKING-STORAGE SECTION.                            | 00034 |                              |                         |          |
| 00035 | 77 SAVED-MONTH PICTURE 99 VALUE ZERO .              | 00035 | REF BY                       | 00110 00144 00146       |          |
| 00036 | 77 WORK-AREA-1 PICTURE X(78) VALUE "MONTH DAY DEPT. | 00036 |                              |                         | PU       |
| 00037 | "RCHASES TYPE COST CUMULATIVE-COSTS "               | 00037 | REF BY                       | 00091                   |          |
| 00038 | 01 STORAGE-TABLE-FOR.                               | 00038 | CONTAINS                     | 117 CHARACTERS          | 20 WORDS |
| 00039 | 02 RECORD-MONTH-NAMES.                              | 00039 |                              |                         |          |
| 00040 | 03 FILLER PIC A(9) VALUE "JANUARY "                 | 00040 | STARTS IN CHARACTER POSITION |                         | 1        |
| 00041 | 03 FILLER PIC A(9) VALUE "FEBRUARY "                | 00041 | STARTS IN CHARACTER POSITION |                         | 10       |
| 00042 | 03 FILLER PIC A(9) VALUE "MARCH "                   | 00042 | STARTS IN CHARACTER POSITION |                         | 19       |
| 00043 | 03 FILLER PIC A(9) VALUE "APRIL "                   | 00043 | STARTS IN CHARACTER POSITION |                         | 28       |
| 00044 | 03 FILLER PIC A(9) VALUE "MAY "                     | 00044 | STARTS IN CHARACTER POSITION |                         | 37       |
| 00045 | 03 FILLER PIC A(9) VALUE "JUNE "                    | 00045 | STARTS IN CHARACTER POSITION |                         | 46       |
| 00046 | 03 FILLER PIC A(9) VALUE "JULY "                    | 00046 | STARTS IN CHARACTER POSITION |                         | 55       |
| 00047 | 03 FILLER PIC A(9) VALUE "AUGUST "                  | 00047 | STARTS IN CHARACTER POSITION |                         | 64       |
| 00048 | 03 FILLER PIC A(9) VALUE "SEPTEMBER"                | 00048 | STARTS IN CHARACTER POSITION |                         | 73       |
| 00049 | 03 FILLER PIC A(9) VALUE "OCTOBER "                 | 00049 | STARTS IN CHARACTER POSITION |                         | 82       |
| 00050 | 03 FILLER PIC A(9) VALUE "NOVEMBER "                | 00050 | STARTS IN CHARACTER POSITION |                         | 91       |
| 00051 | 03 FILLER PIC A(9) VALUE "DECEMBER "                | 00051 | STARTS IN CHARACTER POSITION |                         | 100      |
| 00052 | 03 FILLER PIC A(9) VALUE "MONTH-ERR"                | 00052 | STARTS IN CHARACTER POSITION |                         | 109      |
| 00053 | 02 RECORD-AREA REDEFINES RECORD-MONTH-NAMES .       | 00053 |                              |                         |          |
| 00054 | 03 MONTHNAME PICTURE A(9) OCCURS 13 TIMES .         | 00054 | STARTS IN CHARACTER POSITION |                         | 1        |
|       |                                                     |       | REF BY                       | 00095 00099 00122 00158 |          |
| 00055 | 01 PURCHASE-TYPE-TABLE.                             | 00055 | CONTAINS                     | 12 CHARACTERS           | 2 WORDS  |
| 00056 | 02 VALUES-BELOW.                                    | 00056 |                              |                         |          |
| 00057 | 03 FILLER PIC X(4) VALUE "CASH" .                   | 00057 | STARTS IN CHARACTER POSITION |                         | 1        |
| 00058 | 03 FILLER PIC X(4) VALUE " CR " .                   | 00058 | STARTS IN CHARACTER POSITION |                         | 5        |
| 00059 | 03 FILLER PIC X(4) VALUE " ** " .                   | 00059 | STARTS IN CHARACTER POSITION |                         | 9        |
| 00060 | 02 TX REDEFINES VALUES-BELOW .                      | 00060 |                              |                         |          |
| 00061 | 03 TY PIC X(4) OCCURS 3 TIMES .                     | 00061 | STARTS IN CHARACTER POSITION |                         | 1        |
|       |                                                     |       | REF BY                       | 00104                   |          |
| 00062 | 01 WORK-AREA.                                       | 00062 | CONTAINS                     | 34 CHARACTERS           | 6 WORDS  |
|       |                                                     |       | REF BY                       | 00089                   |          |
| 00063 | 02 MUNTH PIC X(9) .                                 | 00063 | STARTS IN CHARACTER POSITION |                         | 1        |
|       |                                                     |       | REF BY                       | 00158                   |          |
| 00064 | 02 LINE-DESC-1 PIC X(14) VALUE " EXPENDITURES " .   | 00064 | STARTS IN CHARACTER POSITION |                         | 10       |
| 00065 | 02 CONTINUED PIC X(11) VALUE IS SPACE .             | 00065 | STARTS IN CHARACTER POSITION |                         | 24       |
|       |                                                     |       | REF BY                       | 00144 00145             |          |
| 00066 | REPORT SECTION.                                     | 00066 |                              |                         |          |
| 00067 | RD EXPENSE-REPORT                                   | 00067 |                              |                         |          |
| 00068 | WITH CODE KODE                                      | 00068 |                              |                         |          |
| 00069 | CONTROLS ARE FINAL, MONTH, DAY                      | 00069 |                              |                         |          |

| COBOL<br>ALT # | SOURCE LISTING                                          | REF<br>LINE # | COMPILER | COMMENTS                                            |
|----------------|---------------------------------------------------------|---------------|----------|-----------------------------------------------------|
| 00070          | PAGE LIMIT IS 59 LINES                                  | 00070         |          |                                                     |
| 00071          | HEADING 1                                               | 00071         |          |                                                     |
| 00072          | FIRST DETAIL 9                                          | 00072         |          |                                                     |
| 00073          | LAST DETAIL 48                                          | 00073         |          |                                                     |
| 00074          | FOOTING 52                                              | 00074         |          |                                                     |
| 00075          | 01 TYPE IS REPORT HEADING .                             | 00075         |          |                                                     |
| 00076          | 02 LINE NUMBER 10 .                                     | 00076         |          |                                                     |
| 00077          | 03 COLUMN 1 PICTURE A(26) VALUE IS                      | 00077         |          |                                                     |
| 00078          | "BOLT MANUFACTURING COMPANY" .                          | 00078         |          |                                                     |
| 00079          | 03 COLUMN 55 PICTURE A(16) VALUE IS                     | 00079         |          |                                                     |
| 00080          | "PHOENIX DIVISION" .                                    | 00080         |          |                                                     |
| 00081          | 02 LINE NUMBER 13 .                                     | 00081         |          |                                                     |
| 00082          | 03 COLUMN NUMBER 1                                      | 00082         |          |                                                     |
| 00083          | PICTURE IS A(29) VALUE IS                               | 00083         |          |                                                     |
| 00084          | "QUARTERLY EXPENDITURES REPORT" .                       | 00084         |          |                                                     |
| 00085          | 03 COLUMN NUMBER 59                                     | 00085         |          |                                                     |
| 00086          | PICTURE IS A(8) VALUE IS "XYZ DEPT" .                   | 00086         |          |                                                     |
| 00087          | 01 PAGE-HEAD TYPE IS PAGE HEADING .                     | 00087         |          |                                                     |
| 00088          | 02 LINE NUMBER 1                                        | 00088         |          |                                                     |
| 00089          | COLUMN 30 PIC X(34) SOURCE IS WORK-AREA .               | 00089         | REF TO   | 00062                                               |
| 00090          | 02 LINE NUMBER 7                                        | 00090         |          |                                                     |
| 00091          | COLUMN 2 PIC X(78) SOURCE IS WORK-AREA-1 .              | 00091         | REF TO   | 00037                                               |
| 00092          | 01 TYPE CONTROL HEADING MONTH NEXT GROUP PLUS 1 .       | 00092         |          |                                                     |
| 00093          | 02 LINE PLUS 1 .                                        | 00093         |          |                                                     |
| 00094          | 03 COLUMN 2 PIC X(23) VALUE "TRANSACTION JOURNAL FOR" . | 00094         |          |                                                     |
| 00095          | 03 COLUMN 31 PIC A(9) SOURCE MONTHNAME (MONTH) .        | 00095         | REF TO   | 00026 00054                                         |
| 00096          | 01 DETAIL-LINE TYPE IS DETAIL                           | 00096         |          |                                                     |
| 00097          | LINE NUMBER IS PLUS 1 .                                 | 00097         |          |                                                     |
| 00098          | 02 COLUMN 2 GROUP INDICATE PICTURE A(9)                 | 00098         |          |                                                     |
| 00099          | SOURCE IS MONTHNAME OF RECORD-AREA (MONTH) .            | 00099         | REF TO   | 00026 00054                                         |
| 00100          | 02 COLUMN 13 GROUP INDICATE PICTURE 99                  | 00100         |          |                                                     |
| 00101          | SOURCE IS DAY .                                         | 00101         | REF TO   | 00027                                               |
| 00102          | 02 COLUMN 19 PIC XXX SOURCE IS DEPT .                   | 00102         | REF TO   | 00021                                               |
| 00103          | 02 COLUMN 31 PIC Z9 SOURCE IS NO-PURCHASES .            | 00103         | REF TO   | 00023                                               |
| 00104          | 02 COLUMN 40 PIC XXXX SOURCE IS TY (TYPE-PURCHASE) .    | 00104         | REF TO   | 00025 00061                                         |
|                |                                                         | *             | EF       | FORMAT OR RADIX CONVERSION REQUIRED<br>ON SUBSCRIPT |
| 00105          | 02 COLUMN 50 PIC ZZ9.99- SOURCE IS COST .               | 00105         | REF TO   | 00029                                               |
| 00106          | 02 PIC ZZ9.99- SOURCE COST1 .                           | 00106         |          |                                                     |
| 00107          | 01 TYPE CONTROL FOOTING DAY NEXT GROUP PLUS 2 .         | 00107         |          |                                                     |
| 00108          | 02 LINE NUMBER IS PLUS 2 .                              | 00108         |          |                                                     |

| COBOL<br>ALT # | SOURCE LISTING                                         | REF<br>LINE # | COMPILER<br>COMMENTS                    |
|----------------|--------------------------------------------------------|---------------|-----------------------------------------|
| 00109          | 03 COLUMN 2 PIC X(22) VALUE "TRANSACTION TOTALS FOR".  | 00109         |                                         |
| 00110          | 03 COLUMN 24 PIC Z9 SOURCE SAVED-MONTH.                | 00110         | REF TO 00035                            |
| 00111          | 03 COLUMN 26 PIC X VALUE "-".                          | 00111         |                                         |
| 00112          | 03 COLUMN 27 PIC 99 SOURCE DAY.                        | 00112         | REF TO 00027                            |
| 00113          | 03 COLUMN 30 PIC ZZ9 SUM NO-PURCHASES.                 | 00113         | REF TO 00023                            |
| 00114          | 03 MIN COLUMN 49 PIC \$\$\$9.99- SUM COST.             | *             | EF FORMAT OR RADIX CONVERSION REQUIRED. |
| 00115          | 03 COLUMN 60 PIC X(7) VALUE "TO-DATE".                 | 00114         | REF TO 00029                            |
| 00116          | 03 COLUMN 68 PIC \$\$\$9.99- SUM COST1 RESET ON FINAL. | *             | EF FORMAT OR RADIX CONVERSION REQUIRED. |
| 00117          | 02 LINE PLUS 1.                                        | 00115         |                                         |
| 00118          | 03 COLUMN 1 PIC X(78) VALUE ALL "-".                   | 00116         | REF TO 00030                            |
| 00119          | 01 TYPE CONTROL FOOTING MONTH                          | *             | EF FORMAT OR RADIX CONVERSION REQUIRED. |
| 00120          | LINE NUMBER IS 51 NEXT GROUP NEXT PAGE .               | 00117         |                                         |
| 00121          | 02 COLUMN 16 PIC A(14) VALUE "TOTAL COST FOR" .        | 00118         |                                         |
| 00122          | 02 COLUMN 31 PIC A(9) SOURCE MONTHNAME (MONTH) .       | 00119         |                                         |
| 00123          | 02 COLUMN 43 PIC AAA VALUE "WAS" .                     | 00120         |                                         |
| 00124          | 02 INT                                                 | 00121         |                                         |
| 00125          | COLUMN 48 PIC \$\$\$9.99 SUM MIN.                      | 00122         | REF TO 00026 00054                      |
| 00126          | 02 COLUMN 60 PIC X(7) VALUE "TO-DATE".                 | 00123         |                                         |
| 00127          | 02 COLUMN 68 PIC \$\$\$9.99- SUM COST1 RESET ON FINAL. | 00124         |                                         |
| 00128          | 01 TYPE CONTROL FOOTING FINAL LINE NEXT PAGE .         | 00125         |                                         |
| 00129          | 02 COLUMN 16 PIC A(26) VALUE IS                        | 00126         |                                         |
| 00130          | "TOTAL COST FOR QUARTER WAS" .                         | 00127         | REF TO 00030                            |
| 00131          | 02 COLUMN 45 PIC \$\$\$9.99 SUM INT .                  | *             | EF FORMAT OR RADIX CONVERSION REQUIRED. |
| 00132          | 02 COLUMN 55 PIC X(12) VALUE "YEAR-TO-DATE".           | 00128         |                                         |
| 00133          | 02 COLUMN 68 PIC \$\$\$9.99- SUM COST1.                | 00129         |                                         |
| 00134          | 01 TYPE PAGE FOOTING LINE 57 .                         | 00130         |                                         |
| 00135          | 02 COLUMN 60 PIC X(12) VALUE "REPORT-PAGE-" .          | 00131         |                                         |
| 00136          | 02 COLUMN 72 PIC 99 SOURCE PAGE-COUNTER .              | 00132         |                                         |
| 00137          | 01 TYPE REPORT FOOTING LINE PLUS 1                     | 00133         | REF TO 00030                            |
| 00138          | COLUMN 62 PIC X(10) VALUE "END REPORT" .               | *             | EF FORMAT OR RADIX CONVERSION REQUIRED. |
| 00139          | *EJECT                                                 | 00134         |                                         |
|                |                                                        | 00135         |                                         |
|                |                                                        | 00136         |                                         |
|                |                                                        | ***           | WA INTEGER TRUNCATION REQUIRED.         |
|                |                                                        | 00137         |                                         |
|                |                                                        | 00138         |                                         |
|                |                                                        | 00139         |                                         |

| COBOL<br>ALT # | SOURCE LISTING                                             | REF<br>LINE # | COMPILER | COMMENTS                                            |
|----------------|------------------------------------------------------------|---------------|----------|-----------------------------------------------------|
| 00140          | PROCEDURE DIVISION.                                        | 00140         |          |                                                     |
| 00141          | DECLARATIVES.                                              | 00141         |          |                                                     |
| 00142          | PAGE-HEAD-OPTION SECTION. USE BEFORE REPORTING PAGE-HEAD . | U00142        |          |                                                     |
| 00143          | PAGE-CONTINUATION-TEST.                                    | P00143        |          |                                                     |
| 00144          | IF MONTH = SAVED-MONTH, MOVE "(CONTINUED)" TO CONTINUED    | 00144         | REF TO   | 00026 00035 00065                                   |
| 00145          | ELSE MOVE SPACES TO CONTINUED                              | 00145         | REF TO   | 00065                                               |
| 00146          | MOVE MONTH TO SAVED-MONTH .                                | 00146         | REF TO   | 00026 00035                                         |
| 00147          | PAGE-CONTINUATION-EXIT.                                    | P00147        |          |                                                     |
| 00148          | EXIT .                                                     | 00148         |          |                                                     |
| 00149          | END DECLARATIVES.                                          | 00149         |          |                                                     |
| 00150          |                                                            | 00150         |          |                                                     |
| 00151          | OPEN-FILES.                                                | P00151        |          |                                                     |
| 00152          | OPEN INPUT CARD-FILE, OUTPUT REPORT-FILE .                 | 00152         |          |                                                     |
| 00153          | READ CARD-FILE AT END GO TO CLOSE-FILES .                  | P00153        |          |                                                     |
| 00154          | INITIATE EXPENSE-REPORT .                                  | 00154         | REF TO   | 00162                                               |
| 00155          | PROCESS-DATA.                                              | P00155        |          |                                                     |
| 00156          | IF MONTH > 12 OR LESS THAN 01, MOVE 13 TO MONTH .          | 00156         | REF BY   | 00161                                               |
| 00157          | IF TYPE-PURCHASE > 3 OR < 1 MOVE 3 TO TYPE-PURCHASE .      | 00157         | REF TO   | 00026                                               |
| 00158          | MOVE MONTHNAME OF RECORD-AREA (MONTH) TO MUNTH .           | 00157         | REF TO   | 00025                                               |
|                |                                                            | 00158         | REF TO   | 00026 00054 00063                                   |
|                |                                                            | *             | EF       | FORMAT OR RADIX CONVERSION REQUIRED<br>ON SUBSCRIPT |
| 00159          | GENERATE DETAIL-LINE .                                     | 00159         |          |                                                     |
| 00160          | READ CARD-FILE AT END GO TO CLOSE-FILES .                  | P00160        |          |                                                     |
| 00161          | GO TO PROCESS-DATA .                                       | 00161         | REF TO   | 00162                                               |
| 00162          | CLOSE-FILES.                                               | P00162        | REF TO   | 00155                                               |
| 00163          | TERMINATE EXPENSE-REPORT .                                 | 00163         | REF BY   | 00153 00160                                         |
| 00164          | CLOSE CARD-FILE REPORT-FILE .                              | 00164         |          |                                                     |
| 00165          | STOP RUN .                                                 | 00165         |          |                                                     |

\*\*\*\*\* THE ABOVE LISTING CONTAINS 000 ERROR MESSAGES \*\*\*\*\*

\*\*\* THE ABOVE LISTING CONTAINS 001 WARNING MESSAGES \*\*\*

\* THE ABOVE LISTING CONTAINS 007 EFFICIENCY MESSAGES \*

COMPILATION TIME (MIN): ELAP CLOCK= 000.21 PROC= 000.11

00000 OVERFLOW READS 00000 OVERFLOW WRITES 22978 WORDS MEMORY USED 00007 LINKS USED ON \*3 FILE

BOLT MANUFACTURING COMPANY

PHOENIX DIVISION

QUARTERLY EXPENDITURES REPORT

XYZ DEPT

JANUARY EXPENDITURES

| MONTH                       | DAY | DEPT. | PURCHASES | TYPE | COST    | CUMULATIVE-COSTS |
|-----------------------------|-----|-------|-----------|------|---------|------------------|
| TRANSACTION JOURNAL FOR     |     |       | JANUARY   |      |         |                  |
| TRANSACTION TOTALS FOR 1-21 |     |       | 1         |      | \$16.00 | TO-DATE \$16.00  |
| JANUARY                     | 23  | XYZ   | 1         | CR   | 32.00   |                  |
| TRANSACTION TOTALS FOR 1-23 |     |       | 1         |      | \$32.00 | TO-DATE \$48.00  |
| JANUARY                     | 24  | XYZ   | 1         | CR   | 64.00   |                  |
| TRANSACTION TOTALS FOR 1-24 |     |       | 1         |      | \$64.00 | TO-DATE \$112.00 |
| JANUARY                     | 25  | XYZ   | 1         | CASH | 32.00   |                  |
|                             |     | XYZ   | 1         | CASH | 32.00   |                  |
| TRANSACTION TOTALS FOR 1-25 |     |       | 2         |      | \$64.00 | TO-DATE \$176.00 |
| JANUARY                     | 28  | XYZ   | 1         | CR   | 32.00   |                  |
|                             |     | XYZ   | 1         | CR   | 32.00   |                  |
| TRANSACTION TOTALS FOR 1-28 |     |       | 2         |      | \$64.00 | TO-DATE \$240.00 |
| JANUARY                     | 29  | XYZ   | 1         | **   | 16.00   |                  |
|                             |     | XYZ   | 1         | **   | 16.00   |                  |
| TRANSACTION TOTALS FOR 1-29 |     |       | 2         |      | \$32.00 | TO-DATE \$272.00 |
| JANUARY                     | 30  | XYZ   | 1         | CR   | 8.00    |                  |
| TRANSACTION TOTALS FOR 1-30 |     |       | 1         |      | \$8.00  | TO-DATE \$280.00 |
| JANUARY                     | 31  | XYZ   | 1         | CASH | 4.00    |                  |

REPORT-PAGE-01



JANUARY EXPENDITURES (CONTINUED)

| MONTH                       | DAY | DEPT. | PURCHASES | TYPE | COST    | CUMULATIVE-COSTS |
|-----------------------------|-----|-------|-----------|------|---------|------------------|
| JANUARY                     | 31  | XYZ   | 1         | CASH | 4.00-   |                  |
|                             |     | XYZ   | 1         | CASH | 64.00   |                  |
| TRANSACTION TOTALS FOR 1-31 |     |       | 3         |      | \$64.00 | TO-DATE \$344.00 |

---

TOTAL COST FOR JANUARY WAS \$344.00 TO-DATE \$344.00

REPORT-PAGE-02

FEBRUARY EXPENDITURES

| MONTH                            | DAY | DEPT. | PURCHASES | TYPE | COST     | CUMULATIVE-COSTS |
|----------------------------------|-----|-------|-----------|------|----------|------------------|
| TRANSACTION JOURNAL FOR FEBRUARY |     |       |           |      |          |                  |
|                                  |     | XYZ   | 1         | CASH | 2.00     |                  |
| TRANSACTION TOTALS FOR 2-01      |     |       | 2         |      | \$4.00   | TO-DATE \$348.00 |
| -----                            |     |       |           |      |          |                  |
| FEBRUARY                         | 02  | XYZ   | 1         | CR   | 128.00   |                  |
| TRANSACTION TOTALS FOR 2-02      |     |       | 1         |      | \$128.00 | TO-DATE \$476.00 |
| -----                            |     |       |           |      |          |                  |

TOTAL COST FOR FEBRUARY WAS \$132.00 TO-DATE \$476.00

REPORT-PAGE-03

FEBRUARY EXPENDITURES (CONTINUED)

| MONTH                                                     | DAY | DEPT. | PURCHASES | TYPE | COST | CUMULATIVE-COSTS |
|-----------------------------------------------------------|-----|-------|-----------|------|------|------------------|
| TOTAL COST FOR QUARTER WAS \$476.00 YEAR-TO-DATE \$476.00 |     |       |           |      |      |                  |

REPORT-PAGE-04  
END REPORT



## SECTION IX

### FILE ORDERING - SORT AND MERGE

#### CONCEPTS

##### Sorting

Much data processing depends upon the order in which records appear in the files being processed. Such processing often depends upon the order of the records being sequenced according to the values of one or more fields that appear in each of the records. The fields upon which the ordering depends are called the 'keys' of the file.

Since data in its original form seldom occurs in well ordered sequences, a technique, sorting, is provided by which the user can impose the desired ordering upon the records in a file. A sorting procedure manipulates an input file whose records are in an indeterminate sequence and produces an output file containing the same set of data records rearranged into the desired sequence. The number of records in the input file is usually unknown at the inception of the sort procedure and is generally not relevant to the sorting procedure.

##### Merging

Some data processing operations are performed on a group of records that are distributed on several files. If all of those several files are themselves well ordered by the same rules, their contents may be merged into one composite file which is itself well ordered. A merging procedure manipulates two or more well ordered input files and produces an output file containing the total set of data records in one well ordered sequence.

##### Ordering

The sequencing of the output file in a sorting or merging procedure is governed by the values of one or more fields in each of the records being ordered. These fields, the keys, must appear in the same position, relative to the start of the record, in every record being sorted or merged.

The order in which the keys are specified to the sort or merge procedure determines their hierarchical relationship. The first key named, the major key, is the most significant field. Each successive key specified is of decreasing significance until the last key, the most minor key, is reached.

Each key may also be specified as determining an ascending ordering or a descending ordering. Ascending ordering means that those records with lower values of that key will appear in the output file prior to the records with higher values for that key. Descending ordering implies the inverse result.

Ordering is accomplished by comparing, from major to most minor, the corresponding keys of two records until an inequality of value is found. The output order is then determined by the ascending or descending rule which applies to that particular key field. If there is no inequality of value in any corresponding pair of keys, the ordering is determined by the sort or merge procedure.

### Program Organization

The COBOL language contains two verbs which initiate ordering procedures, SORT and MERGE. Several additional language features are associated with both of these verbs. The RELEASE verb may be used with the execution of a SORT verb and the RETURN verb may be used with the execution of SORT and MERGE verbs. A special form of the SELECT sentence in the FILE-CONTROL paragraph of the Environment Division is associated with the intermediate working files of the sort procedure and the implicit working file of the merge procedure. In addition, those files are described by a special type of file description (SD rather than FD) in the Data Division.

The SORT verb invokes the execution of a set of sorting procedures contained within the standard software library. These procedures operate with the COBOL object program to perform the sort. During the execution of the sorting function, the object program and sorting procedures combine in the organization pictured in Figure 9-1. The user can include several SORT statements in the source program. If several SORT statements are present, they are completely independent of each other.

The MERGE verb invokes the execution of a set of merging procedures contained within the standard software library. These procedures operate with the COBOL object program to perform the merge. During the execution of the merging function, the object program and merging procedures combine in the organization pictured in Figure 9-2. The user can include several MERGE statements in the source program. If several MERGE statements are present, they are completely independent of each other.

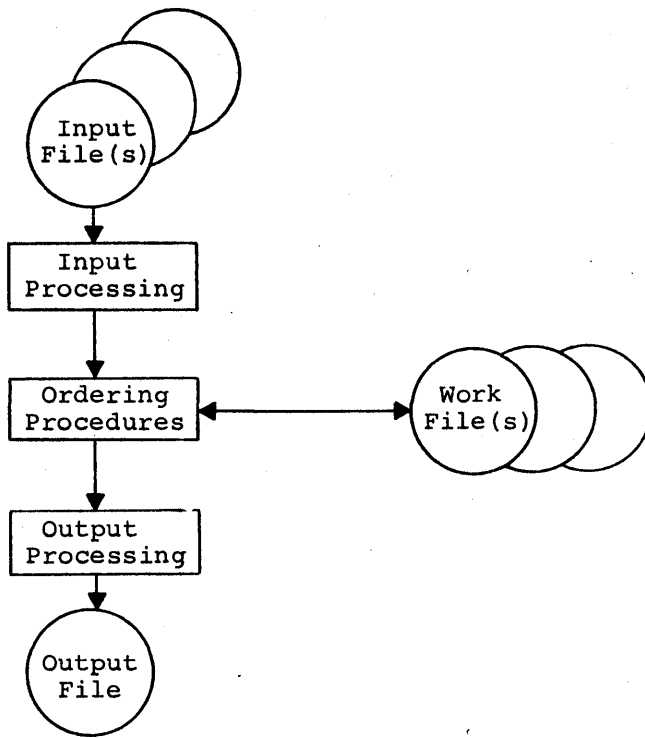


Figure 9-1. Sort Program Organization

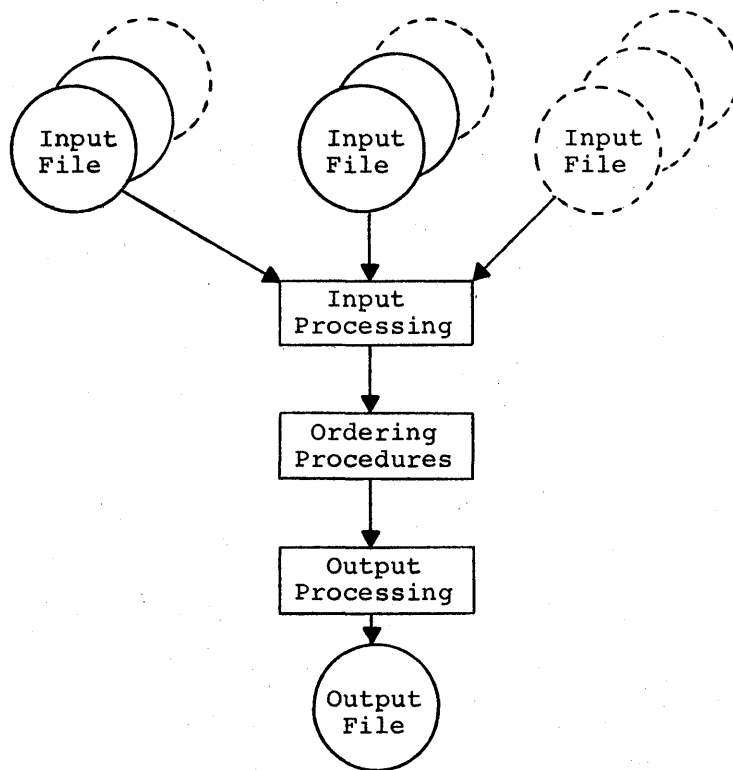


Figure 9-2. Merge Program Organization

## SORT STATEMENT

The purpose of the SORT statement is to invoke the execution of a sorting procedure.

### The Sort File

The definition of the sort file serves two purposes:

1. It is the vehicle for defining the working files associated with the sorting procedure.
2. It is the location where all the keys for the ordering are described.

The former role is required by both the syntax rules of the SORT statement and the sorting procedures. The size of the data file being sorted cannot be determined from within the sorting procedure prior to the inception of the procedure. Therefore, the sort must be prepared to manipulate an input file containing many times the number of records that can be held in memory. The sorting procedures handle such files by distributing subsets of the input data upon working files and then collating these intermediate sets into the final output file. The file space required by this procedure must be allocated by the user.

Sort file allocation occurs on two levels; within the source program, and at the system level. At the source program level, assignment is made in the FILE-CONTROL paragraph with the ASSIGN phrase. One or more file-codes are specified, each of them denoting one working file. The first file-code specified must not appear in any other ASSIGN phrase. None of these file-codes should have 'S' as the first letter. This list of file-codes defines the set of user collation files.

In addition to the set of user collation files specified in the SELECT sentence, the sorting procedures supplied from the system library presume that any of the reserved set of 16 file-codes S1, S2, ..., S16 denote system collation files. If any files have been allocated with these file-codes, they will be used for collation in any and all sorts which are executed within the activity. The use of these 16 file-codes for any other purpose is not recommended.

At the system level, the user must provide peripheral assignment cards for the sort's working files. The file-codes for which assignments are provided may be any combination of the members of the set of user collation file-codes and the members of the set of system collation file-codes. There is no need to make peripheral assignments for all, or for any, of the user collation file-codes. The sorting procedure checks for allocation before attempting to use any of those files. In any case, a sorting procedure will utilize no more than 16 collation files. With the exceptions predicated on device type as described below, user declared collation files will be utilized before system collation files.

The sort working files may be assigned to either tape or random-access devices or to a combination of both devices. If all the files are assigned to tape, at least three files must be assigned. Under certain conditions, it is possible to include input and output tape files within the set of working files. It is possible, although inefficient, to execute a sort with a total of only three tapes assigned to the activity.

If the working files are assigned to random-access devices, only one file need be assigned. For efficiency, the total random-access file space assigned should be sufficient to contain the entire input data file plus an overhead factor of ten percent. If the size of the input data file exceeds the random-access space assigned, the sorting procedures will attempt recovery action, but efficiency will suffer. It is advisable to use more than one file-code when assigning random-access working file space in order to improve activity allocation at run time.

If the sort working files are assigned to a combination of tape and random-access devices, the sorting procedure works with the random-access devices. The tape files will not be utilized unless it is necessary to recover from a collation file overflow condition caused by the amount of input data exceeding the random-access space allocation. This preference for working with the random-access devices will be maintained even if the tape files are part of the set of user collation files.

### Sort Key Declarations

The second purpose of the sort file is to provide a vehicle for the definition of all the sort keys which determine the record ordering. This definition is accomplished in a special type of file description in the File Section of the Data Division.

The definition is prefaced with the level indicator SD. Each key data-name associated with a SORT statement must be defined within the sort file-name description referred to by the SORT statement. The keys are listed in the SORT statement in order, from most significant (major key) to least significant (most minor key), with the word 'ASCENDING' or 'DESCENDING' preceding key data-names as appropriate. Key comparison coding is generated by the COBOL compiler, rather than by the sorting procedure, on the basis of the key declarations in the SORT statement.

When more than one record description entry appears in a sort file description, the key data items need be described in only one of the record description entries. Each key data item must occur in every data record of the sort file. It must have the same relative position and actual format in all records. The PICTURE and USAGE of a given key data item must be the same in all records in the sort file. If a key item is synchronized or justified, it must be identically synchronized or justified in all records in the sort file. The key data item descriptions must not contain an OCCURS clause or be subordinate to entries containing an OCCURS clause. A key data item may not be described with USAGE COMPUTATIONAL-4. Keys must be data items that do not require subscripting or indexing.



## VARIABLE-LENGTH RECORDS

Although key items themselves may not be of variable length, the records within the sort file may be of variable length. Each record must be large enough to contain the entire set of keys described in the SORT statement. Variable-length records may occur as the result of two separate syntactical usages:

1. The sort file and the input file referenced by the USING option both have two or more record descriptions of different sizes (variable size 01 descriptions), or both are specified as variable-length record files in the APPLY phrase.
2. The sort file has a record containing an OCCURS...DEPENDING clause.

In the first case, multiple record descriptions or explicit variable-length record specification, no special handling of the file is necessary to facilitate sorting, assuming that all of the position and format rules are followed. The first record description in the sort file determines the 'dominant record size' parameter for the sort. If an input procedure is used, the record types must be differentiated by separate RELEASE statements for each record size.

In the second case, an OCCURS...DEPENDING option, the file being sorted requires special handling. The simpler, but inefficient, method is to use an input procedure. This technique will ensure that all records are fully expanded during input.

When using the input procedure technique, a key field may fall after an OCCURS item in a record, provided the basic rules on relative positions and formats within records are followed.

A more efficient technique for processing the OCCURS...DEPENDING files utilizes the USING/GIVING option. The following special steps are taken:

1. The sort file is declared as a variable-length record file in the APPLY phrase.
2. The OCCURS...DEPENDING clause is not used in the SD description.
3. The record description in the SD description is started with the item: FILLER PIC X(6). This entry allows for the existence of the occurs control word which appears on each record. The comparison coding generated by the compiler is thereby properly aligned with the described key fields.
4. The SD description is padded out with a filler item to achieve an efficient dominant record length depending upon the characteristics of the file.
5. Since the relative position of the key might shift from record to record, no key field may follow any of the OCCURS items within the records being sorted.

When this technique is used, the USING and GIVING phrases must both be specified, since there is no way for the sort process to expand the record into its full form for the output procedure. Therefore, any processing using the result of the sort requires another pass over the file. However, this may be more efficient than the sorting of the fully expanded but partially 'empty' records.

#### DOMINANT RECORD LENGTH

If the data being sorted is in variable-length record format, the sort procedure must also be given a dominant record size. In the sort procedure, memory is divided into fixed-length packets that are used to handle record input and output. In a fixed-length record sort, the packets are made equal to the record size. Thus, there is no wasted space in memory during input and output processing. However, in a variable-length record sort, the packets are made equal to the dominant record size. Records which are equal to or less than the dominant record size fit into one packet. Records which are larger than the dominant record size are divided into as many packets as necessary to contain the record.

When records deviate from the dominant record size, space in each packet is wasted. Such waste reduces sorting efficiency. The dominant record size should be chosen to minimize this waste space in memory and minimize the number of packets used for any one record. The basis for this practice is that each division of a record adds to the overhead cost of record handling. For example, suppose the following record sizes and frequencies are present:

| <u>Size</u> | <u>Frequency</u> |
|-------------|------------------|
| 15 words    | 50 percent       |
| 17 words    | 20 percent       |
| 30 words    | 30 percent       |

The optimum dominant record size would be 17 words. That value would result in the least wasted space and fewest record divisions when the file is being processed.

The dominant record size is set equal to the size of the first record described in the sort file description entry (SD) in the Data Division.

#### Sort Key Evaluation

When the values of a key in a pair of sort file records are compared, one value is found to be greater than, equal to, or less than the other according to the rules given under the Relation Condition paragraph in Section XII. The key comparison determines the order of the records in the sort output.

1. If all corresponding key values of a pair of sort file records are identical, the record that appeared first in the input to the sorting procedure will precede the other in the output.

2. If any key values of a pair of sort file records are unequal, the output sequence is based upon the most significant key item for which inequality is found.
  - a. If that key item is governed by the ASCENDING option, the record with the lower value will precede the other in the output.
  - b. If the item is governed by the DESCENDING option, the record with the higher value will precede the other in the output.

### Sort Equal Key Procedures

The COBOL sort interface subroutine permits COBOL programs to contain user-supplied procedures for analysis, summarization, and/or conditional deletion of records with equal key values that may be encountered during sorting.

The label .CSEQK, a global symbol in the COBOL sort interface subroutine, is associated with a word through which the entry point address of an equal key procedure may be communicated to the sort process. The address of the equal key procedure entry point must be placed in the lower half of that word without disturbing the contents of the upper half. The additional coding required in the COBOL program in order to use this facility consists of the following four elements, all of which must be written in assembly language (GMAP):

1. Coding to set the equal key procedure address into .CSEQK.
2. Coding to change the address of the equal key procedure, if more than one such procedure is used.
3. Coding to reset .CSEQK to zero, whenever further use of equal key procedures is not desired.
4. Coding for the equal key procedure itself.

### ENGAGEMENT OF EQUAL KEY PROCEDURES

If an equal key procedure is to be operative during the execution of a particular SORT statement, then at some time prior to the execution of that SORT statement the entry point address of the appropriate equal key procedure must be placed in the lower half of the word identified by the global symbol .CSEQK, without disturbing the contents of the upper half of that word. The address of the equal key procedure appearing in .CSEQK is not changed by the sorting process. Thus, unless the content of .CSEQK is explicitly changed by the user's coding, the same equal key procedure will be operative during all subsequent COBOL SORT statement executions in the run unit.

### CHANGING EQUAL KEY PROCEDURES

An alternate equal key procedure may be designated for use in conjunction with subsequent SORT statement executions by changing the address in location .CSEQK. This may only be accomplished between SORT statement executions; after the completion of one SORT statement and before the execution of the next.

## DISENGAGEMENT OF EQUAL KEY PROCEDURES

Resetting the lower half of location .CSEQK terminates further use of equal key procedures. This may only be accomplished between SORT executions, and not during the execution of a SORT statement. Equal key procedures may subsequently be re-engaged, as described above.

## SORT EQUAL KEY RECORD PROCESSING

Whenever two records having equal key values are encountered in the sorting process, and an operative equal key procedure has been designated, control is transferred to the designated procedure with index registers six and seven set to refer to the two records involved. The equal key procedure must then determine which record should be deleted. It may choose to delete neither. Information may also be transferred to the preserved record from the one to be deleted, but the contents of the key fields must not be changed, nor may the program reference any portion of a record that extends beyond the area contained within the dominant record size. After processing the records, the equal key procedure must report its decision to the sort process. The applicable programming conventions are described in the User Squeeze Coding paragraph in the Sort/Merge Program manual.

## Sort Input Processing

A choice must be made between having the sorting process handle the input processing of the file being sorted or having the user's program specify the input processing procedures. In most cases the former technique is more efficient but the latter may be necessary in order to accomplish selective editing of the input records. If such editing would result in the deletion of a significant portion of the input file, then the input procedure technique is more appropriate.

## USING OPTION

If the USING option is specified, the sorting process will handle the input file processing. The file specified by file-name-2 must not be in an open state when the SORT statement is executed. File-name-2 must have a file description (FD) entry in the Data Division. The sort file and the file referenced in the USING phrase are, in a sense, alternative descriptions of the same set of data. Therefore, all file level properties (such as APPLY SYSTEM STANDARD) must be identical for both files. In addition to the file explicitly referenced in the USING phrase, the sorting procedure will presume that any of the reserved set of 16 file-codes SA, SB, ..., SP denote system input files. If any of these files have been allocated in a sorting activity, they will unconditionally be used as input files in every sort process in the activity. Input is taken as the entire content of the USING file and then as the contents of SA, SB, ..., SP, in order.

## INPUT PROCEDURE OPTION

When the INPUT PROCEDURE option is specified, the user is responsible for all the input processing for the sorting process. An input procedure must consist of one or more sections. Since the input procedure is invoked by the sorting procedure via the same techniques used in the PERFORM statement execution, the structure of the input procedure must follow the same basic rules as a set of sections which are the object of a PERFORM statement. Control must not be passed to the input procedure except through the execution of a SORT statement. The input procedure may contain any procedures needed to select, create, or modify records for input to the sorting process. Three general restrictions apply to the procedural statements within the input procedure:

1. An input procedure must not contain a SORT statement, a MERGE statement, or a RETURN statement.
2. The input procedure must not contain transfers of control to points outside the input procedure. This means that the ALTER, GO TO, and PERFORM statements in the input procedure must not refer to procedure-names outside the input procedure. COBOL statements are allowed that will cause an implied transfer of control to USE procedures.
3. The remainder of the Procedure Division must not contain transfers of control to points inside the input procedure. This means that ALTER, GO TO, and PERFORM statements in the remainder of the Procedure Division must not refer to procedure-names within the input procedure.

### RELEASE Statement

The RELEASE statement is used to transfer logical input records from an input procedure to the initial phase of a sorting operation. The RELEASE statement may appear only in an input procedure and every input procedure must contain at least one RELEASE statement. Refer to the COBOL Reference Manual for specific format conventions.

The record-name must be the name of a record defined within a sort file; that is, a file described with an SD level indicator. If the sort file description contains more than one record description, and if the record descriptions define records of different sizes, a separate RELEASE statement must be specified for each record size. If the FROM phrase is used, the contents of 'identifier' must be the name of a data item in working-storage or of an input record area. If the format of identifier is different from that of the record-name, moving takes place according to the rules specified for the MOVE statement without the CORRESPONDING option. The information in the sort record area is no longer available, but the information in the identifier area is available. It is illegal to use the same name for both the record-name and the identifier.

After the RELEASE statement is executed, the contents of record-name are no longer available to the COBOL procedure. The execution of a RELEASE statement causes the contents of record-name (after the contents of identifier have been moved to it in the FROM phrase) to be made available to the initial phase of the sort process. When control passes from the input procedure, the sort file consists of all those records which were placed in it by the execution of RELEASE statements. No OPEN, READ, WRITE, or CLOSE statements may be given for the sort file.

## Sort Output Processing

A choice must be made between having the sorting process handle the output processing of the newly sorted file or having the user's program specify the output processing procedures. The choice is not as easily made as in the case of the input procedure since it involves the assessment between the efficiency of giving the sorting process more memory to work with and the total system efficiency of embedding the processing of the output file within an output procedure. Any requirement for a copy of the file for later processing by other procedures may also be a consideration.

The output file cannot be assigned to SYSOUT or REMOTE.

### GIVING OPTION

If the GIVING option is specified, the sorting process will handle the output file processing. The file specified by file-name-3 must not be in an open state when the SORT statement is executed. File-name-3 must have a file description (FD) entry in the Data Division. The file-name-2 and file-name-3 are, in a sense, alternative descriptions of the same set of data. Therefore, all file level properties (such as APPLY SYSTEM STANDARD) must be identical for both files. If both the USING and GIVING options are specified, they may refer to the same file-name or to different file-names. In the special case in which USING and GIVING refer to the same file-name, the sort procedure will not use the GIVING file as a collation file. Otherwise, the sort procedure will use a GIVING file as a collation tape file unless inhibited by the ELECT SORT OPTIONS phrase (Field-7) in the SPECIAL-NAMES paragraph or by the collation being performed on random-access media.

### OUTPUT PROCEDURE OPTION

When the OUTPUT PROCEDURE option is specified, the final output file processing for the sorting process is the responsibility of the program. An output procedure must consist of one or more sections. Since the output procedure is invoked by the sorting procedure via the same techniques used in the execution of a PERFORM statement, the structure of the output procedure must follow the same basic rules as a set of sections which are the object of a PERFORM statement. Control must not be passed to the output procedure except through the execution of a SORT statement. The output procedure may contain any procedures needed to select, modify, or copy the records which are being returned from the sorting process. Three general restrictions apply to the procedural statements within the output procedure:

1. An output procedure must not contain a SORT statement, a MERGE statement, or a RELEASE statement.
2. The output procedure must not contain transfers of control to points outside the output procedure; i.e., ALTER, GO TO, and PERFORM statements in the output procedure are not permitted to refer to procedure-names outside the output procedure. COBOL statements are allowed that will cause an implied transfer of control to USE procedures.
3. The remainder of the Procedure Division must not contain transfers of control to points inside the output procedure; i.e., ALTER, GO TO, and PERFORM statements in the remainder of the Procedure Division are not permitted to refer to procedure-names within the output procedure.

## RETURN Statement

The RETURN statement is used to obtain logical output records from a sort operation and to transfer them to an output procedure. The RETURN statement may appear only in an output procedure, and every output procedure must contain at least one RETURN statement. Refer to the COBOL Reference Manual for specific format conventions.

File-name must be described with an SD level indicator and must be the same file-name that was referenced in the SORT statement currently being executed.

The INTO phrase may be used only when the file referred to by file-name contains just one type of record. The identifier must be the name of a data item in working-storage or of an output record area. If the format of the identifier differs from that of the input record, moving is performed according to the rules specified for the MOVE statement without the CORRESPONDING option. When the INTO phrase is used, 'file-name RECORD' is still available in the file-name record area.

The execution of the RETURN statement causes the next record in sorted order (according to the keys listed in the SORT statement) to be made available for processing in the record area associated with the sort file. No OPEN, READ, WRITE, or CLOSE statements may be given for the sort file.

After the contents of the sort file are exhausted, the next execution of the RETURN statement will result in the execution of the imperative-statement in the AT END phrase.

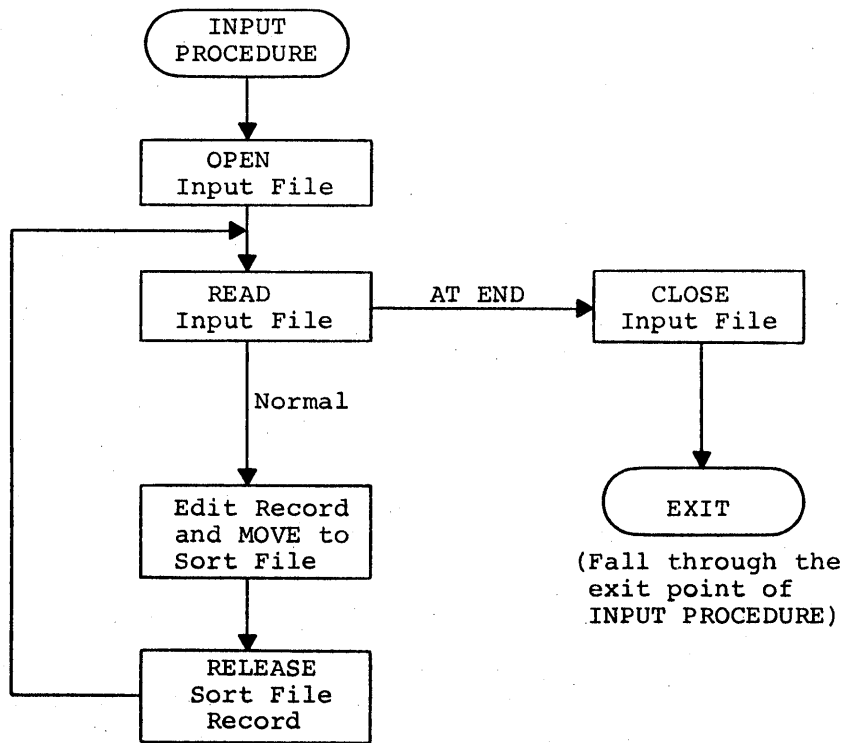
After execution of the AT END imperative-statement, no RETURN statement may be executed within the current output procedure. The results of such an error are unpredictable.

## SORT OPERATIONAL CONSIDERATIONS

### Flow of Control

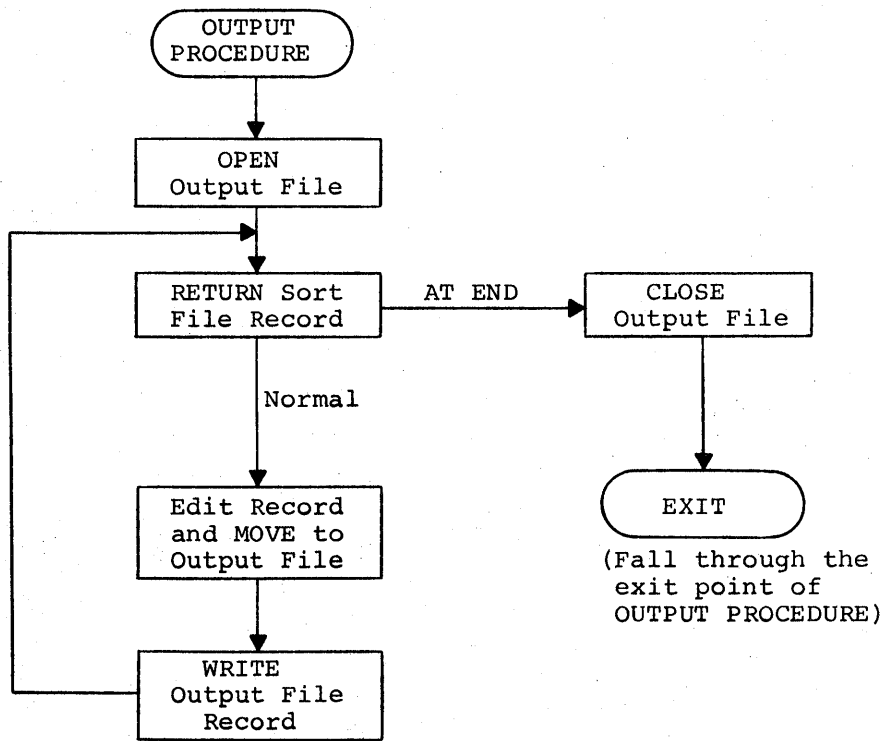
Any sequence of procedural statements may be executed before or after the SORT statement is executed. When the SORT statement is executed, the sorting process receives control.

If an input procedure has been specified, the sort transfers control to the input procedure as soon as it is ready to start processing records. The input procedure opens the input file and reads the first record. Each time that a record is ready for the sort file, the input procedure executes a RELEASE statement, causing the sort to place the record in the sort file. Control then passes to the statement following the RELEASE statement. The input procedure continues reading and releasing until all the input records have been given to the sort, at which time the input procedure closes the input file and allows control to pass to its exit point, thereby returning control to the sorting process. A simple input procedure might be organized as follows:





The sorting process orders the records, up to the point of determining which record goes first in the final output sequence. If an output procedure has been specified, the sorting process at this point transfers control to the output procedure. The output procedure opens its output file, if any, and then obtains the first record in the final sequence by executing a RETURN statement. When the output procedure has disposed of the first record, it returns the next, and thus continues returning records and processing them. After the last record has been returned, the sorting process causes control to pass to the AT END phrase the next time a RETURN statement is executed. The output procedure closes its output file, if any, and allows control to pass to its exit point. The sorting process then terminates its own procedures. Control then passes to the statement following the SORT statement. A simple output procedure might be organized as follows:



In effect, the sequence of events just described applies also when the USING or GIVING option is used, except that the input or output procedure becomes implicit, rather than specified in detail by the user.

### Reserved File-Codes

File-codes beginning with the character 'S' have special meaning to the sorting process. (Such file-codes include S1, S2, ..., SA, SB, ..., SZ.) In any COBOL program that utilizes the SORT statement, the file-codes specified in all SELECT sentences must not begin with the character 'S'.

## Implicit File Assignments

### INPUT FILES

The sort process assumes that the file-codes in the series SA, SB, ..., SP designate only input files. If, during a sort execution, any file has been allocated using one of these file-codes, that file will unconditionally be used as input to the sort process. In no case, however, will more than a total of 16 files be used as input to a sort process.

### OUTPUT FILE

The sort process assumes that the file-code SZ designates only an output file.

### COLLATION FILE-CODES

The sort process assumes that the file-codes in the series S1, S2, ..., SØ designate only collation working files. If, during a sort execution, any file has been allocated using one of these file-codes, that file will unconditionally be used as a collation file.

Unless inhibited by Field-7 of the ELECT phrase in the SPECIAL-NAMES paragraph or by the execution of a mass storage sort, the output file-code is added to the list of collation file-codes. If the user has provided an output procedure, the file-code SZ is unconditionally added to the list of collation file-codes. In no case, however, will more than 16 files be used for the sort collation process.

### BORROWED FILE-CODES

If the sort process is allowed to borrow collation space from the operating system, it assumes that file-codes in the series SQ, SR, ..., SV are restricted to this purpose. Explicit assignment of any of these restricted file-codes during a sort process may cause unpredictable results.

### Collation File Manipulation

If all collation files are assigned to tape devices, the tape sort is engaged. If the output file is to be used as a collation file, it must be assigned to a tape device. If the sort has from three to five collation tapes, it manipulates them in the polyphase mode. At the end of each collation phase, two of the collation tapes must be rewound before the program can continue. If the sort has from six to 16 collation tapes, it manipulates them in the standby mode. At least two of the tapes are rewound during the collation process, so that no phase must wait for tapes to finish rewinding in order to begin.

If a mass storage random-access device is allocated for one or more of the collation files, the mass storage sort is engaged. The allocated collation files are manipulated as if they were one conglomerate random-access area. The allocation of several files on different channels increases input-output overlap, however. The output file cannot be used for collation.

If, during sorting, the mass storage area is exhausted, the sort procedure will attempt two levels of recovery. The first level of spill recovery attempts to borrow more random file space from the operating system. This attempt is made via newly created files. If the user has already allocated 16 mass storage files for collation, the sort will not attempt the first spill recovery level. When the area borrowed for recovery is exhausted, the sort attempts to borrow more random file space on the same newly created files. If all requests for more links are denied, the sort attempts the second level of spill recovery on tapes. If the user has allocated collation files on tape devices, they will be used. If no files, or less than three files, have been allocated, the sort attempts to borrow tapes from the operating system. If unsuccessful, the sort execution aborts. Otherwise, the sort purges the current string to mass storage and reconfigures itself as a tape sort. The remainder of the input is distributed on tapes. The strings on tapes are collated to one string. The strings on mass storage are collated to one string. These two strings are then merged to the output file.

If both tape and random-access devices are allocated, the sort will select the random-access mode of operation. If necessary, the tape files are utilized to recover from random file overflow.

The disposition of collation files is controlled by the file-code assignment specified for the sort file. The following discussion presents several alternative file-code strategies, and assumes knowledge of the functions of the \$ TAPE and \$ NTAPE control cards (refer to the Control Cards manual).

A basic requirement is that the tape sort must have at least three collation tapes. Whatever file-code strategy is chosen, this requirement must be satisfied. Sources of collation tapes are as follows:

1. A \$ NTAPE card specifying file-code S1 may provide some or all of the collation tapes. Depending upon how many tapes the \$ NTAPE card provides, file-codes in the series S1, S2, ..., will be available for the sort.
2. As described above, the GIVING option may normally provide one collation tape to the tape sort.
3. Additional collation tapes may be provided via extra file-codes specified in the sort file SELECT sentence.

It is recommended that the first (or only) file-code mentioned in the sort file SELECT sentence not be represented by a \$ TAPE card when the object program is executed. The \$ NTAPE card approach is considered the standard means of providing collation tapes.

The file-code strategy recommended for a tape oriented sort file is, therefore, to assign a single file-code (thereby satisfying COBOL syntax rules); to omit the \$ TAPE card for that file-code when the object program is executed; and to supply instead a \$ NTAPE card specifying file-code S1 and at least three tapes.

Multiple file-codes in the sort file SELECT sentence may be desired for various reasons:

1. If an output procedure produces a magnetic tape output file, the file-code assigned to that file may be the second specified in the sort file SELECT sentence (the first file-code should not be represented by a \$ TAPE card during program execution). This technique puts the tape at the sort's disposal throughout the collation activity, but guarantees that the tape will be free when the output procedure is engaged.
2. If one or more tape files are totally processed before or after the sort procedure, the tapes are therefore available as scratch tapes throughout the sort execution. Listing their file-codes in the sort file SELECT sentence leads to enhanced collating efficiency. Such file-codes would also be specified in the SELECT sentences for the files to which they are assigned.
3. If the \$ NTAPE card is not desired, multiple file-codes in the sort file SELECT sentence can be employed to give the user full control over the allocation of collation tapes via \$ TAPE cards.

If the basic requirement for three collation tapes is satisfied, \$ TAPE cards for the extra sort file file-codes may optionally be omitted during program execution. If \$ TAPE cards are provided, the sort process assumes such tapes are available for collation purposes.

The USING and GIVING file-codes must not be specified in the ASSIGN phrase of the SELECT sentence for the sort file.

When sort file-codes are represented by \$ TAPE cards during program execution, each such file-code counts as only one tape toward the sort's minimum requirement for three collation tapes even if the \$ TAPE card calls for alternating tape handlers. Similarly, the GIVING file-code (when applicable) counts as only a single collation tape.

If the sort file is associated with disk devices, the same basic techniques apply, with two major exceptions. First: Only one disk file need be provided for the collation area. This file must be a random-access file and should define sufficient space to hold all the records from the input file plus an overhead factor of ten percent. Second: In no case should the output file be associated with the collation file space. If the GIVING option is used, file-name-3 will be explicitly excluded from the collation file space. If an output procedure is used, no attempt should be made to force sort's use of the output file during collation. The file will not be free during the execution and, if used as collation, would cause unpredictable results.

### Sort Configuration

The memory requirements for a sort execution depend on the file description and the amount of user coding. A sort of a system standard file, having little user coding and utilizing mass storage or three tape collation files, will execute in the 16,000 words of memory allocated as the loader default assumption.

The sort process requires an appropriate collation file configuration before it begins execution. The minimum configuration for a mass storage sort is one file. Normally, more files are used. The total mass storage area should be adequate to hold all the input file plus a ten percent overhead factor. If the amount of data to be sorted is small enough to allow the sort to be memory contained, execution may begin with only one assigned random link. If the sort is memory contained, the file is not accessed. If the sort is not memory contained, the sort control program tests for the availability of sufficient mass storage to contain the tournament entries. If insufficient space has been allocated, the sort attempts to borrow space from the operating system. If unsuccessful, the program aborts.

The sort program can also be executed with a minimum of three tape files. Such an execution requires the multiple use of tapes for input, collation, and output. The sort process distributes information in an orderly manner on the collation tapes. Since there must be a free tape for the succeeding collation pass, at least one tape is not written on during the input process. This tape is unconditionally the last tape in the specified series of collation files. The series is composed of those allocated files specified in the ASSIGN phrase followed by the allocated system files (S1, S2, ...).

In the same manner, a free tape must be available to receive the output of the last collation pass. This tape is unconditionally the first tape in the specified series of collation files. If the sort is allowed to collate on the specified output tape, that tape is always assumed to be first in the collation series.

Thus, it is possible to use one tape for both output and collation, another for collation only, and another for both input and collation. The \$ TAPE control cards could be configured as follows:

```
$ TAPE SZ,X1R output file
$ TAPE S1,X2R
$ TAPE SA,X3R input file
$ TAPE S2,X3R
```

Since file S2 and file SA have the same logical unit designator, only three tape units will be used.

#### Memory Assignment for Sort

Normally, the sorting procedure utilizes the memory area designated as open in the slave program prefix information. This information is provided by the General Loader during program execution. The size of the area for any given program may be defined with the \$ LIMITS card. Conflicts will arise, however, when another system program (e.g., I-D-S) is trying to utilize the same area. In that circumstance, the user must have some means of dividing the free area of memory among the system programs. Memory is easily divided by removing the area assigned to the sorting procedure from the free area designated by the General Loader.

The sorting procedure has been made sensitive to the locations of two Labeled Common storage areas: .SMA and .SMC. During preliminary processing, the location of these two areas is checked by the sorting procedures. If the distance between the two areas exceeds four words, it is assumed that the sorting procedures will operate entirely within the space between the two areas.

The position and size of the two Labeled Common areas are controlled by the \$ USE card. For example, the following deck setup defines the Labeled Common storage areas:

```
$ IDENT
$ LOWLOAD
$ USE .SMA/1/,.SMB/50000/,.SMC/1/
```

Object decks or compilations follow this point.

This method allows the user to reorganize memory at load time without recompiling any programs.

### Dynamic Resource Allocation

If a tape oriented sort has been engaged, the use of Field-5 and Field-9 of the ELECT phrase makes the sorting procedure responsible for the dynamic allocation of more memory and collation tapes to the current activity. The sort first borrows more memory from the operating system, up to the limit set by the user. If there is more than enough memory to provide double buffering of the collation tapes at the largest possible block size, the sort borrows more tapes from the operating system for collation. The sort will borrow tapes until one of the following occurs:

1. The number of available tapes is exhausted.
2. The limit set by the user is reached.
3. The use of another tape would cause a decrease in collation block size.
4. The maximum of 16 collation tapes is reached.

Sort Examples

The following example illustrates a basic SORT program:

```
.
.
ENVIRONMENT DIVISION.
FILE-CONTROL.
 SELECT INPUT-FILE ASSIGN TO AB.
 SELECT SORT-FILE ASSIGN TO CD.
 SELECT OUTPUT-FILE ASSIGN TO EF.
I-O-CONTROL. APPLY SYSTEM STANDARD ON
 INPUT-FILE, SORT-FILE, OUTPUT-FILE.
DATA DIVISION.
FILE SECTION.
FD INPUT-FILE...
 01...
.
.
SD SORT-FILE...
 01...
.
.
FD OUTPUT-FILE...
 01...
.
.
PROCEDURE DIVISION.
SORT-CALL. SORT SORT-FILE ON ...USING
 INPUT-FILE GIVING OUTPUT-FILE.
STOP RUN.
```

Note that these files may have multiple record types and sizes, provided the sort file and USING file have the same records, the sort file and GIVING file have the same records, and key descriptions and positions are equivalent for all record types.

The control cards for program execution could be as follows:

```
.
.
$ EXECUTE
$ LIMITS ...
$ TAPE AB,A1D,,,,INPUT-LABEL
$ TAPE EF,B1S,,,,OUTPUT-LABEL
$ NTAPE S1,C,2 (Two or more tapes required in this example)
```

The control cards could also be as follows:

```
.
.
$ EXECUTE
$ LIMITS ...
$ TAPE AB,A1D,,,,INPUT-LABEL
$ TAPE EF,B1S,,,,OUTPUT-LABEL
$ FILE S1,C1R,nnnnR
```

Another SORT feature entails the use of an output procedure to deliver a report (on any suitable device) rather than an output tape as such:

```
.
.
ENVIRONMENT DIVISION.
FILE-CONTROL.
 SELECT INPUT-FILE ASSIGN TO GH.
 SELECT SORT-FILE ASSIGN TO IJ, KL.
 SELECT REPORT-OUTPUT ASSIGN TO KL FOR LISTING.
DATA DIVISION.
FILE SECTION.
FD INPUT-FILE...
 01...
.
.
SD SORT-FILE...
 01...
.
.
FD REPORT-OUTPUT; REPORT IS XYZ...
.
.
WORKING-STORAGE SECTION.
.
.
REPORT SECTION.
RD XYZ...
 01 DETAIL-LINE; TYPE DE...
.
.
PROCEDURE DIVISION.
SORT-CALL SECTION.
DRIVER. SORT SORT-FILE ON ... USING INPUT-FILE
 OUTPUT PROCEDURE IS EDIT. STOP RUN.
EDIT SECTION.
STARTUP. OPEN REPORT-OUTPUT; INITIATE XYZ.
LOOP. RETURN SORT-FILE RECORD; AT END GO TO QUIT.
.
.
 GENERATE DETAIL-LINE; GO TO LOOP.
QUIT. TERMINATE XYZ; CLOSE REPORT-OUTPUT.
```

Since file-code KL is used for the output procedure's output file and is also mentioned in the sort file SELECT sentence, it must be associated with a magnetic tape during program execution. The control cards should therefore be as follows (assuming the sort is to use only three collation tapes):

```
.
.
$ EXECUTE
$ LIMITS ...
$ TAPE GH,A1D,,,,INPUT-LABEL
$ TAPE KL,B1S,,,,OUTPUT-LABEL
$ NTAPE S1,C,2
```

If KL were not mentioned in the sort file SELECT sentence, its control card could specify any suitable output medium such as \$ TAPE or \$ SYSOUT. Except in the latter case, the sort activity would presumably be followed by a Bulk Media Conversion activity to print the report.



## MERGE STATEMENT

The purpose of the MERGE statement is to invoke the execution of a merging procedure.

### The Merge File

The definition of the merge file serves only to provide a location where all the keys for the ordering are described. There is no application of the merge file which corresponds to the working file status of the sort file. However, COBOL syntax rules require that an appropriate SELECT sentence be specified for the merge file. A file assignment control card should not be included for the file-code assigned to the merge file.

### Merge Key Declarations

The definition of the merge key fields is accomplished in a special type of file description in the File Section of the Data Division. The definition is prefaced with the level indicator SD. Each key data-name associated with a MERGE statement must be defined within the merge file-name description referred to by the MERGE statement. The keys are listed in the MERGE statement in order, from most significant (major key) to least significant (most minor key), with the word 'ASCENDING' or 'DESCENDING' preceding key data-names as appropriate. Key comparison coding is generated by the COBOL compiler, rather than by the merging procedure, on the basis of the key declarations in the MERGE statement.

Each key data item must occur in every data record of the merge file. It must have the same relative position and actual format in all records. The PICTURE and USAGE of a given key item must be the same in all records in the merge file. If a key item is synchronized or justified, it must be identically synchronized or justified in all records in the merge file. A key data item may not be described with USAGE COMPUTATIONAL-4. Keys must be data items which do not require subscripting or indexing.

### VARIABLE-LENGTH RECORDS

Although key items themselves may not be of variable length, the records within the merge file may be of variable length. Each record must be large enough to contain the entire set of keys described in the MERGE statement. Variable-length records may occur as the result of two separate syntactical usages:

1. The merge file and the input file referenced by the USING option both have two or more record descriptions of different sizes (variable size 01 descriptions), or both are specified as variable-length record files in the APPLY phrase.
2. The input file has a record containing an OCCURS...DEPENDING clause.

In the first case, multiple record descriptions, no special handling of the file is necessary to facilitate merging, assuming that all of the position and format rules are followed.

In the second case, an OCCURS...DEPENDING option, the files being merged require special handling utilizing the GIVING option. The following special steps are taken:

1. The merge file is declared as a variable-length record file in the APPLY phrase.
2. The OCCURS...DEPENDING clause is not used in the SD description.
3. The record description in the SD description is started with the item: FILLER PIC X(6). This entry allows for the existence of the occurs control word which appears on each record. The comparison coding generated by the compiler is thereby properly aligned with the described key fields.
4. Since the relative position of the key might shift from record to record, no key field may follow any of the OCCURS items within the records being sorted.

When this technique is used, the GIVING phrase must be specified, since there is no way for the merge process to expand the record into its full form for the output procedure. Therefore, any processing using the result of the merge requires another pass over the file.

#### Merge Key Evaluation

When the values of a key in a pair of merge file records are compared, one value is found to be greater than, equal to, or less than the other according to the rules given under the Relation Condition paragraph in Section XII. The key comparison determines the order of the records in the merge output.

1. If all corresponding key values of a pair of merge file records are identical, the record that appeared first in the input to the merging procedure will precede the other in the output.
2. If any key values of a pair of merge file records are unequal, the output sequence is based upon the most significant key item for which inequality is found.
  - a. If that key item is governed by the ASCENDING option, the record with the lower value will precede the other in the output.
  - b. If the item is governed by the DESCENDING option, the record with the higher value will precede the other in the output.

#### Merge Equal Key Procedures

The COBOL merge interface subroutine permits COBOL programs to contain user-supplied procedures for analysis, summarization, and/or conditional deletion of records with equal key values that may be encountered during merging.

The label `.CMEQK`, a global symbol in the COBOL merge interface subroutine, is associated with a word through which the entry point address of an equal key procedure may be communicated to the merge process. The address of the equal key procedure entry point must be placed in the lower half of that word without disturbing the contents of the upper half. The additional coding required in the COBOL program in order to use this facility consists of the following four elements, all of which must be written in assembly language (GMAP):

1. Coding to set the equal key procedure address into `.CMEQK`.
2. Coding to change the address of the equal key procedure, if more than one such procedure is used.
3. Coding to reset `.CMEQK` to zero, whenever further use of equal key procedures is not desired.
4. Coding for the equal key procedure itself.

#### ENGAGEMENT OF EQUAL KEY PROCEDURES

If an equal key procedure is to be operative during the execution of a particular MERGE statement, then at some time prior to the execution of that MERGE statement the entry point address of the appropriate equal key procedure must be placed in the lower half of the word identified by the global symbol `.CMEQK`, without disturbing the contents of the upper half of that word. The address of the equal key procedure appearing in `.CMEQK` is not changed by the merging process. Thus, unless the content of `.CMEQK` is explicitly changed by the user's coding, the same equal key procedure will be operative during all subsequent COBOL MERGE statement executions in the run unit.

#### CHANGING EQUAL KEY PROCEDURES

An alternate equal key procedure may be designated for use in conjunction with subsequent MERGE statement executions by changing the address in location `.CMEQK`. This may only be accomplished between MERGE statement executions; after the completion of one MERGE statement and before the execution of the next.

#### DISENGAGEMENT OF EQUAL KEY PROCEDURES

Resetting the lower half of location `.CMEQK` terminates further use of equal key procedures. This may only be accomplished between MERGE executions, and not during the execution of a MERGE statement. Equal key procedures may subsequently be re-engaged, as described above.

## MERGE EQUAL KEY RECORD PROCESSING

Whenever two records having equal key values are encountered in the merging process, and an operative equal key procedure has been designated, control is transferred to the designated procedure with index registers six and seven set to refer to the two records involved. The equal key procedure must then determine which record should be deleted. It may choose to delete neither. Information may also be transferred to the preserved record from the one to be deleted, but the contents of the key fields must not be changed, nor may the program reference any portion of a record that extends beyond the area contained within the dominant record size. After processing the records, the equal key procedure must report its decision to the merge process. The applicable programming conventions are described in the User Squeeze Coding paragraph in the Sort/Merge Program manual.

### Merge Input Processing

When a merging procedure is used, no options are available for handling input files. The merging process will handle all of the processing of input files. The USING phrase must be specified in the MERGE statement and must identify at least two file-names. The referenced input files must not be in an open state when the MERGE statement is executed. All of the specified file-names must have file description (FD) entries in the Data Division. The merge file and the files referenced in the USING phrase are, in a sense, alternative descriptions of the same set of data. Therefore, all file level properties (such as APPLY SYSTEM STANDARD) must be identical for all of the files. In addition to the files explicitly referenced in the USING phrase, the merging procedure will presume that any of the reserved set of 16 file-codes SA, SB, ..., SP denote system input files. If any of these files have been allocated in a merging activity, they will unconditionally be used as input files in every merge process in the activity. The hierarchy of input files is first the series of files in the USING phrase followed by the files SA, SB, ..., SP, in order.

### Merge Output Processing

A choice must be made between having the merging process handle the output processing of the newly merged file or having the user's program specify the output processing procedures.

### GIVING OPTION

If the GIVING option is specified, the merging process will handle the output file processing. The file specified by file-name-5 must not be in an open state when the MERGE statement is executed. File-name-5 must have a file description (FD) entry in the Data Division. File-name-5 and the files specified in the USING phrase are, in a sense, alternative descriptions of the same set of data. Therefore, all file level properties (such as APPLY SYSTEM STANDARD) must be identical for both files.

## OUTPUT PROCEDURE OPTION

When the OUTPUT PROCEDURE option is specified, the final output file processing for the merging process is the responsibility of the program. An output procedure must consist of one or more sections. Since the output procedure is invoked by the merging process via the same techniques used in the execution of a PERFORM statement, the structure of the output procedure must follow the same basic rules as a set of sections which are the object of a PERFORM statement. Control must not be passed to the output procedure except through the execution of a MERGE statement. The output procedure may contain any procedures needed to select, modify, or copy the records which are being returned from the merging process. Three general restrictions apply to the procedural statements within the output procedure:

1. An output procedure must not contain a MERGE statement, a SORT statement, or a RELEASE statement.
2. The output procedure must not contain transfers of control to points outside the output procedure; i.e., ALTER, GO TO, and PERFORM statements in the output procedure are not permitted to refer to procedure-names outside the output procedure. COBOL statements are allowed that will cause an implied transfer of control to USE procedures.
3. The remainder of the Procedure Division must not contain transfers of control to points inside the output procedure; i.e., ALTER, GO TO, and PERFORM statements in the remainder of the Procedure Division are not permitted to refer to procedure-names within the output procedure.

### RETURN Statement

The RETURN statement is used to obtain logical output records from a merge operation and to transfer them to an output procedure. The RETURN statement may appear only in an output procedure, and every output procedure must contain at least one RETURN statement. Refer to the COBOL Reference Manual for specific format conventions.

File-name must be described with an SD level indicator and must be the same file-name that was referenced in the MERGE statement currently being executed.

The INTO phrase may be used only when the file referred to by file-name contains just one type of record. The identifier must be the name of a data item in working-storage or of an output record area. If the format of the identifier differs from that of the input record, moving is performed according to the rules specified for the MOVE statement without the CORRESPONDING option. When the INTO phrase is used, 'file-name RECORD' is still available in the file-name record area.

The execution of the RETURN statement causes the next record in merged order (according to the keys listed in the MERGE statement) to be made available for processing in the record area associated with the merge file. No OPEN, READ, WRITE, or CLOSE statements may be given for the merge file.

After the contents of the merge file are exhausted, the next execution of the RETURN statement will result in the execution of the imperative-statement in the AT END phrase.

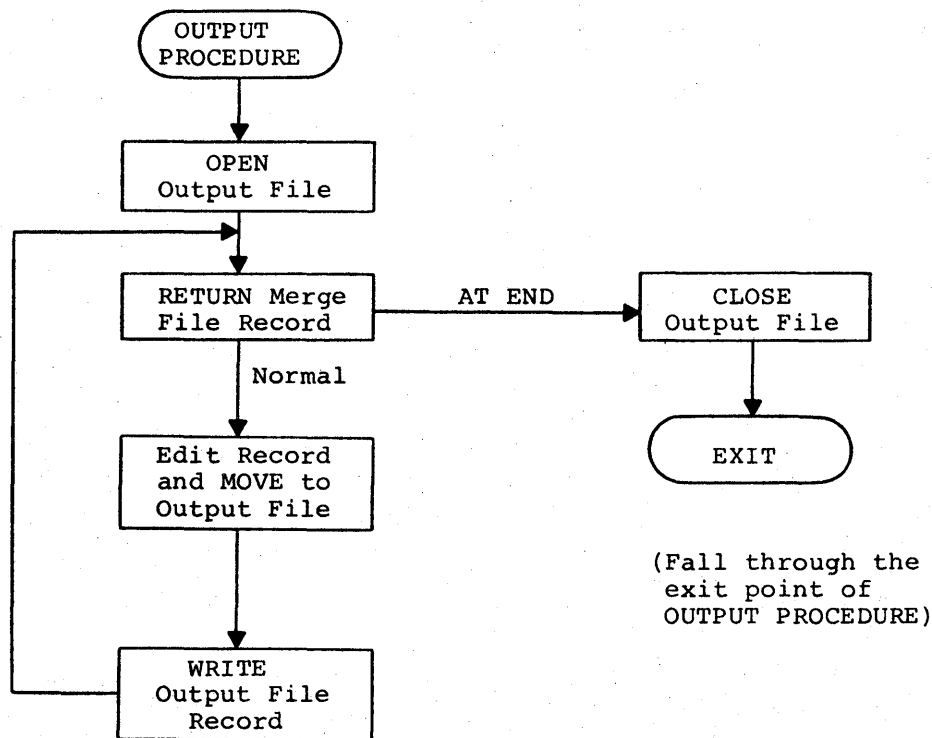
After execution of the AT END imperative-statement, no RETURN statement may be executed within the current output procedure. The results of such an error are unpredictable.

### MERGE OPERATIONAL CONSIDERATIONS

#### Flow of Control

Any sequence of procedural statements may be executed before or after the MERGE statement is executed. When the MERGE statement is executed, the merging process receives control.

The merging process performs initial housekeeping, up to the point of determining which record is placed first in the final output sequence. If an output procedure has been specified, the merging process at this point transfers control to the output procedure. The output procedure opens its output file, if any, and then obtains the first record in the final sequence by executing a RETURN statement. When the output procedure has disposed of the first record, it returns the next, and thus continues returning records and processing them. After the last record has been returned, the merging process causes control to pass to the AT END phrase the next time a RETURN statement is executed. The output procedure closes its output file and allows control to pass to its exit point. The merging process then terminates its own procedures. Control then passes to the statement following the MERGE statement. A simple output procedure might be organized as follows:



In effect, the sequence of events just described applies also when the GIVING option is used, except that the output procedure becomes implicit, rather than specified in detail by the user.

### Reserved File-Codes

File-codes beginning with the character 'S' have special meaning to the merging process. (Such file-codes include SA, SB, ..., SZ.) In any COBOL program that utilizes the MERGE statement, the file-codes specified in all SELECT sentences must not begin with the character 'S'.

### Implicit File Assignments

#### INPUT FILES

The merge process assumes that the file-codes in the series SA, SB, ..., SP designate only input files. If, during a merge execution, any file has been allocated using one of these file-codes, that file will unconditionally be used as input to the merge process. In no case, however, will more than a total of 16 files be used as input to a merge process.

#### OUTPUT FILE

The merge process assumes that the file-code SZ designates only an output file.

### Merge Configuration

The memory requirements for a merge execution depend on the file descriptions and the amount of user coding. A merge of two system standard files, having little user coding, will execute in the 16,000 words of memory allocated as the loader default assumption.

### Memory Assignment for Merge

Normally, the merging procedure utilizes the memory area designated as open in the slave program prefix information. This information is provided by the General Loader during program execution. The size of the area for any given program may be defined with the \$ LIMITS card. Conflicts will arise, however, when another system program (e.g., I-D-S) is trying to utilize the same area. In that circumstance, the user must have some means of dividing the free area of memory among the system programs. Memory is easily divided by removing the area assigned to the merging procedure from the free area designated by the General Loader.

The merging procedure has been made sensitive to the locations of two Labeled Common storage areas: .SMA and .SMC. During preliminary processing, the location of these two areas is checked by the merging procedure. If the distance between the two areas exceeds four words, it is assumed that the merging procedure will operate entirely within the space between the two areas.

The position and size of the two Labeled Common areas are controlled by the \$ USE control card. For example, the following deck setup defines the Labeled Common storage areas:

```
$ IDENT
$ LOWLOAD
$ USE .SMA/1/,.SMB/50000/,.SMC/1/
```

Object decks or compilations follow this point.

This method allows the user to reorganize memory at load time without reassembling any programs.

Merge Examples

The following example illustrates a basic MERGE program:

```

.
.
ENVIRONMENT DIVISION.
FILE-CONTROL.
 SELECT INPUT-FILE-1 ASSIGN TO AB.
 SELECT INPUT-FILE-2 ASSIGN TO CD.
 SELECT MERGE-FILE ASSIGN TO EF.
 SELECT OUTPUT-FILE ASSIGN TO GH.
I-O-CONTROL. APPLY SYSTEM STANDARD ON INPUT-FILE-1,
 INPUT-FILE-2, MERGE-FILE, OUTPUT-FILE.
DATA DIVISION.
FILE SECTION.
FD INPUT-FILE-1...
 01...
.
.
FD INPUT-FILE-2...
 01...
.
.
SD MERGE-FILE...
 01...
.
.
FD OUTPUT-FILE...
 01...
.
.
PROCEDURE DIVISION.
MERGE-CALL. MERGE MERGE-FILE ON ... USING INPUT-FILE-1, INPUT-FILE-2,
 GIVING OUTPUT-FILE.

```

Note that these files may have multiple record types and sizes, provided the merge file and USING file have the same records, the merge file and GIVING file have the same records, and key descriptions and positions are equivalent for all record types.



The control cards for program execution could be as follows:

```
.
.
$ EXECUTE
$ LIMITS ...
$ TAPE AB,A1D,,,,INPUT-LABEL-1
$ TAPE CD,A2D,,,,INPUT-LABEL-2
$ TAPE GH,B1S,,,,OUTPUT-LABEL
```

Another MERGE feature entails the use of an output procedure to deliver a report (on any suitable device) rather than an output tape as such:

```
.
.
ENVIRONMENT DIVISION.
FILE-CONTROL.
 SELECT INPUT-FILE-1 ASSIGN TO AB.
 SELECT INPUT-FILE-2 ASSIGN TO CD.
 SELECT MERGE-FILE ASSIGN TO EF.
 SELECT REPORT-OUTPUT ASSIGN TO GH FOR LISTING.
DATA DIVISION.
FILE SECTION.
FD INPUT-FILE-1...
 01...
.
.
FD INPUT-FILE-2...
 01...
.
.
SD MERGE-FILE...
 01...
.
.
FD REPORT-OUTPUT, REPORT IS XYZ...
.
.
WORKING-STORAGE SECTION.
.
.
REPORT SECTION.
RD XYZ...
 01 DETAIL-LINE, TYPE DE...
.
.
PROCEDURE DIVISION.
MERGE-CALL SECTION.
DRIVER. MERGE MERGE-FILE ON ... USING
 INPUT-FILE-1, INPUT-FILE-2, OUTPUT
 PROCEDURE IS EDIT.
 STOP RUN.
EDIT SECTION.
STARTUP. OPEN REPORT-OUTPUT, INITIATE XYZ.
LOOP. RETURN MERGE-FILE RECORD, AT END
 GO TO QUIT.
.
.
 GENERATE DETAIL-LINE, GO TO LOOP.
QUIT. TERMINATE XYZ, CLOSE REPORT-OUTPUT.
```

The control cards for program execution could be as follows:

```

.
.
$ EXECUTE
$ LIMITS ...
$ TAPE AB,A1D,,,,INPUT-LABEL-1
$ TAPE CD,A2D,,,,INPUT-LABEL-2
$ TAPE GH,B1S,,,,OUTPUT-LABEL
```

The file control card for GH could specify any suitable output medium such as tape, disk, or SYSOUT.

#### SORT-MERGE ELECTIVE OPTIONS

The ELECT SORT OPTIONS phrase in the SPECIAL-NAMES paragraph of the Environment Division is a compiler-directing phrase that is used to exercise greater control over the sorting or merging process.

The file-names in the ELECT phrase must be described in sort-merge file description entries in the Data Division with the special level indicator SD. A given file-name must not appear in more than one ELECT phrase.

Twelve options (fields) are provided, which correspond to the 12 fields of the ELECT macro of the freestanding sort-merge system. The order and relative position of the fields must be specified according to the definitions of the field numbers listed in the SPECIAL-NAMES paragraph of the COBOL Reference Manual. When the ELECT option is used, each field must be specified according to field description or as a null field. In all cases, the first field must be specified. For additional information, refer to the Sort/Merge Program manual.



## SECTION X

### DATA MOVEMENT PROCEDURES

#### MOVE STATEMENT

In any MOVE statement, two or more receiving items may be specified. The effect is equivalent to a series of separate MOVE statements, each having identifier-1 as the sending item, with identifier-2, identifier-3, etc., respectively, as receiving items.

A MOVE statement whose sending and receiving data items are both elementary items is considered to be an elementary MOVE statement. For elementary MOVE statement rules, elementary data items are classified into the following categories:

- N - Numeric. This includes any item whose PICTURE consists solely of characters from the set 9, S, V, and P. It also includes the figurative constant ZERO and any numeric literal.
- NE - Numeric Edited. An item has at least one of the following:
  1. An editing clause (e.g., BLANK WHEN ZERO).
  2. A PICTURE containing any of the numeric editing characters Z \* \$ , . + - CR and DB.
  3. A PICTURE containing at least one insertion character B, and not containing any of the characters A or X.
- AE - Alphanumeric Edited. An item whose PICTURE contains at least one insertion character B, and at least one character X.
- AN - Alphanumeric. An item whose PICTURE contains only characters from the set A, X, 9 treated as if all were the character X. It also includes nonnumeric literals and the figurative constants except for ZERO and SPACES.
- AB - Alabetic. An item whose PICTURE consists entirely of the character A. It also includes the figurative constant SPACES.

The following rules apply to an elementary MOVE between the categories defined above:

- a. It is illegal to move an NE, AE, the figurative constant SPACE, or an AB item to an N or NE item.
- b. It is illegal to MOVE an N, the figurative constant ZERO, or an NE item to an AB item.

- c. It is illegal to MOVE an N item whose implicit decimal point is not immediately to the right of the least significant digit to an AN or AE item.
- d. A data item described with USAGE DISPLAY-2 can be moved only to a data item described with USAGE DISPLAY or USAGE DISPLAY-2. When a DISPLAY-2 data item is a receiving item, the sending item must be described with USAGE DISPLAY or DISPLAY-2 or be a literal or a figurative constant.

An elementary MOVE statement may be legal or illegal, depending upon the categories to which the respective data items belong. The following chart shows all of the possible elementary MOVE statements.

| Category of Sending Data Item |            | Category of Receiving Data Item |                                     |                                                      |
|-------------------------------|------------|---------------------------------|-------------------------------------|------------------------------------------------------|
|                               |            | Alphabetic                      | Alphanumeric Edited<br>Alphanumeric | Numeric Integer<br>Num. Noninteger<br>Numeric Edited |
| Alphabetic                    |            | Legal/1                         | Legal/1                             | Illegal                                              |
| Alphanumeric                  |            | Legal/3                         | Legal/1                             | Legal/2                                              |
| Alphanumeric Edited           |            | Legal/3                         | Legal/1                             | Illegal                                              |
| Numeric                       | Integer    | Illegal                         | Legal/1                             | Legal/2                                              |
|                               | Noninteger | Illegal                         | Illegal                             | Legal/2                                              |
| Numeric Edited                |            | Illegal                         | Legal/1                             | Illegal                                              |

The results obtained with legal elementary MOVE statements are given below:

Case 1: When an alphanumeric edited or alphanumeric item is a receiving item, alignment and any necessary space-filling takes place as defined under the JUSTIFIED clause and the Standard Alignment Rules. If the size of the sending item is greater than the size of the receiving item, the excess characters are truncated on the right after the receiving item is filled. If the sending item is described as signed numeric, the operational sign will not be moved.

Case 2: When a numeric or numeric edited item is the receiving item, alignment by decimal point and any necessary zero-filling takes place as defined under the Standard Alignment Rules, except where zeros are replaced because of editing requirements.

- a. When a signed numeric item is the receiving item, the sign of the sending item is placed in the receiving item. Conversion of the representation of the sign takes place as necessary. If the sending item is unsigned, a positive sign is generated for the receiving item.
- b. When an unsigned numeric item is the receiving item, the absolute value of the sending item is moved and no operational sign is generated for the receiving item.
- c. When a data item described as alphanumeric is the sending item, data is moved as if the sending item were described as an unsigned numeric integer.

Case 3: When a receiving field is described as alphabetic, justification and any necessary space-filling takes place as defined under the JUSTIFIED clause and the Standard Alignment Rules. If the size of the sending item is greater than the size of the receiving item, the excess characters are truncated on the right after the receiving item is filled.

NOTE: Any necessary conversion of data from one form of internal representation to another (binary to decimal, numeric mode to alphanumeric mode, etc.) takes place during the move, along with any specified editing in the receiving item such as suppressing zeros with blanks, or inserting a dollar sign, commas, a decimal point, etc.

#### EXAMPLES OF MOVE STATEMENTS

The data items referenced in the following examples are assumed to have the data images shown below (V appears in the values of some numeric items to illustrate assumed decimal point alignment; it does not actually appear in data items in the object program):

| <u>Data-Name</u> | <u>PICTURE</u> |
|------------------|----------------|
| A                | 999999V        |
| B                | 9999V99        |
| E                | X(10)          |
| D                | X(6)           |
| C                | 999V9          |
| F                | X(10)          |

| <u>Statement</u>     | <u>Sending Item Contents</u> | <u>Result</u>          |
|----------------------|------------------------------|------------------------|
| MOVE 1000 TO B.      | 1000                         | 1000V00                |
| MOVE 300 TO B.       | 300                          | 0300V00                |
| MOVE 1.235 TO B.     | 1.235                        | 0001V23                |
| MOVE C TO B.         | 024V5                        | 0024V50                |
| MOVE B TO C.         | 1234V56                      | 234V5                  |
| MOVE F TO E.         | AA1673BBCC                   | AA1673BBCC             |
| MOVE D TO F.         | AA1673                       | AA1673 <del>BBCC</del> |
| MOVE E TO D.         | AA1673BBCC                   | AA1673                 |
| MOVE "123456" to D.  | 123456                       | 123456                 |
| MOVE A TO D.         | 007340V                      | 007340                 |
| MOVE D TO A.         | 653000                       | 653000V                |
| MOVE ALL "Z" TO E.   |                              | ZZZZZZZZZZ             |
| MOVE ALL "XY" TO E.  |                              | XYXYXYXYXY             |
| MOVE ALL "XYZ" TO E. |                              | XYZXYZXYZX             |
| MOVE ZEROS TO C.     |                              | 000V0                  |
| MOVE SPACES TO E.    |                              | <del>XXXXXXXXXX</del>  |

Some examples of the editing features of the MOVE statement are given below:

| <u>Sending Item</u> |                              | <u>Receiving Item</u> |                                |
|---------------------|------------------------------|-----------------------|--------------------------------|
| <u>Data Image</u>   | <u>Value of Sending Item</u> | <u>Data Image</u>     | <u>Value of Receiving Item</u> |
| 9999V99             | 567891                       | 9999V99               | 567891                         |
| 9999V99             | 567891                       | 9999V9                | 56789                          |
| 9V9                 | 78                           | 999V99                | 00780                          |
| XXX                 | M8N                          | XXXXX                 | M8N <del>BB</del>              |
| 99V99               | 6789                         | 999.99                | 067.89                         |
| AAAAAA              | WARREN                       | AAA                   | WAR                            |
| 99V99               | 6789                         | \$ZZZ9.99             | \$ <del>BB</del> 67.89         |

(~~B~~ = space or blank)

The following examples illustrate the results of various MOVE statements (V is shown to illustrate the assumed decimal point position):

```
77 B PICTURE 9999V99.
77 C PICTURE 999V9.
77 D PICTURE X(6).
```

| <u>MOVE</u> | <u>Sending Value</u> | <u>Receiving Value</u> |
|-------------|----------------------|------------------------|
| 1000 TO B   | 1000                 | 1000V00                |
| 300 TO B    | 300                  | 0300V00                |
| 1.235 TO B  | 1.235                | 0001V23                |
| C TO B      | 024V5                | 0024V50                |
| D TO B      | 123456               | 3456V00                |

| <u>Sending Item</u> |              | <u>Receiving Item</u> |                        |
|---------------------|--------------|-----------------------|------------------------|
| <u>PICTURE</u>      | <u>Value</u> | <u>PICTURE</u>        | <u>Resulting Value</u> |
| 9(5)                | 45678        | \$ZZ,ZZ9.99           | \$45,678.00            |
| 9(3)V99             | 456.78       | \$ZZ,ZZ9.99           | \$ 456.78              |
| 9(3)V99             | 000.67       | \$ZZ,ZZ9.99           | \$ 0.67                |
| 9(3)V99             | 000.04       | \$ZZ,ZZZ.99           | \$ .04                 |
| 9(5)                | 00000        | \$ZZ,ZZZ.ZZ           |                        |
| V9(5)               | .12345       | \$ZZ,ZZ9.99           | \$ 0.12                |
| 9(5)                | 12345        | \$**,**9.99           | \$12,345.00            |
| 9(5)                | 67890        | \$\$\$,\$\$9.99       | \$67,890.00            |
| 9(3)V99             | 678.90       | \$\$\$,\$\$9.99       | \$678.90               |
| 9(5)                | 00000        | \$\$\$,\$\$9.99       | \$0.00                 |
| V9(5)               | .67890       | \$\$\$,\$\$9.99       | \$0.67                 |
| S9(5)V              | -56789.      | -ZZZZ9.99             | -56789.00              |
| S9(5)               | +56789       | -ZZZZ9.99             | 56789.00               |
| S9(5)               | -56789       | +ZZZZ9.99             | -56789.00              |
| S9(5)               | +56789       | +ZZZZ9.99             | +56789.00              |
| S99V9(3)            | -56.789      | -----9.99             | -56.78                 |
| S9(5)               | -00567       | ZZZZZ.99-             | 567.00-                |
| S9(5)               | -56789       | \$\$\$\$\$.99CR       | \$56789.00CR           |
| S9(5)               | +56789       | \$\$\$\$\$.99CR       | \$56789.00             |

A MOVE statement is considered to be nonelementary if the sending data item and/or the receiving data item is a group item. A nonelementary MOVE statement produces the same effect as if the sending and receiving data items were simple alphanumeric items.

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

In the Extended Instruction Set (EIS) mode, movement of alphanumeric data (such as #, [ ) to a numeric field may result in the abort message ILLEGAL EIS DATA. This type of MOVE is accepted in the non-EIS mode even though it violates COBOL rules.

Cascading moves in which the sending and receiving character positions occur within the same word-pair produce unpredictable results.

#### MOVE CORRESPONDING STATEMENT

The CORRESPONDING option causes selected items subordinate to identifier-1 to be moved to corresponding items subordinate to identifier-2. The selected items are moved individually; editing, format conversion, fill, or truncation take place as appropriate, item by item.

In a MOVE CORRESPONDING statement, identifier-1, identifier-2, ..., must be group items. For each possible pair of items subordinate to identifier-1 and identifier-2, a correspondence exists if the following rules are satisfied:

1. The respective identifiers are the same, including all qualifiers up to (but not including) identifier-1 and identifier-2.
2. One or both of the items in the pair are elementary. (One may be a group item.)



3. Any data-names that are subordinate to identifier-1 or identifier-2 and which have REDEFINES, RENAMES, OCCURS, or USAGE IS INDEX clauses are ignored, as well as any data-names that are subordinate to the data-names containing REDEFINES, RENAMES, OCCURS, or USAGE IS INDEX clauses.

Identifier-1 or identifier-2 may have REDEFINES or OCCURS clauses and may be subordinate to data-names that have REDEFINES or OCCURS clauses.

4. Neither item has level-numbers 66, 77, or 88 or is described with the USAGE IS INDEX clause.

For each corresponding pair of items, the effect of a MOVE CORRESPONDING statement is the same as if a separate MOVE statement had been written instead. For example, consider the statement MOVE CORRESPONDING ABLE TO BAKER (I), where the respective data descriptions are as follows:

|    |                  |    |                           |
|----|------------------|----|---------------------------|
| 03 | ABLE...          | 02 | BAKER; OCCURS 10 TIMES... |
| 04 | P...             | 05 | P                         |
| 04 | Q...             | 05 | Q                         |
| 04 | R REDEFINES Q... | 05 | R                         |
| 05 | S...             | 06 | S                         |

The effect is the same as if the following statements had been written:

```
MOVE P OF ABLE TO P OF BAKER (I)
MOVE Q OF ABLE TO Q OF BAKER (I)
```

Note that R is not moved (because neither R is elementary). S is not moved because S of ABLE is subordinate to a group item within ABLE that is described with the REDEFINES clause.

A correspondence never exists for FILLER items.

Since a MOVE CORRESPONDING statement expands into a series of separate moves, the CORRESPONDING option should never be used for record-to-record or group-to-group moves where the sending and receiving groups have the same descriptions; a simple MOVE statement (without the CORRESPONDING option) produces more efficient object program coding. The COBOL compiler may abort when the MOVE CORRESPONDING statement is used in a program with a very large number of Data Division entries or with very long record descriptions in which 'correspondences' would exist. The abort may be avoided in subsequent compilations by increasing the memory limits. If the memory limits cannot be increased, the abort can be circumvented by replacing the MOVE CORRESPONDING statement with elementary MOVE statements to accomplish the desired functions.

EXAMPLES OF MOVE CORRESPONDING STATEMENTS

Examples of MOVE CORRESPONDING statements that reflect the hierarchy of level structure are given below:

```
01 A
 02 Z
 02 B
 02 C
 03 D
 03 E
 04 F
 04 G
 05 P
 05 Q
 02 H
 03 I
 03 J
 02 K
 02 L
```

```
01 X
 02 Y
 03 L
 04 M
 04 N
 03 C
 04 D
 04 E
 05 F
 05 G
 06 U
 06 V
 03 H
 03 B
 02 Z
 02 Q
```

MOVE CORRESPONDING A TO X is equivalent to

MOVE Z OF A TO Z OF X

MOVE CORRESPONDING A TO Y is equivalent to

```
MOVE B OF A TO B OF Y (elementary)
MOVE D OF A TO D OF Y (elementary)
MOVE F OF A TO F OF Y (elementary)
MOVE L OF A TO L OF Y (group)
MOVE H OF A TO H OF Y (group)
```

MOVE CORRESPONDING Y TO A is equivalent, with sending and receiving items reversed, to

MOVE CORRESPONDING A TO Y

## REPLACING PHRASE (EXAMINE STATEMENT)

The REPLACING phrase of the EXAMINE statement is used to modify the value of an item, by replacing certain characters in the original value with a new character.

When the REPLACING phrase is used (Format 2), character replacement proceeds according to the options specified:

| <u>Option</u> | <u>Substitute Literal-4 For</u>                                                                     |
|---------------|-----------------------------------------------------------------------------------------------------|
| ALL           | Each occurrence of literal-3.                                                                       |
| FIRST         | The first occurrence of literal-3.                                                                  |
| LEADING       | All occurrences of literal-3 prior to the leftmost occurrence of any other character.               |
| UNTIL FIRST   | All occurrences of other characters prior to (but not including) the first occurrence of literal-3. |

For nonnumeric data items, examination starts at the leftmost character and proceeds to the right. Each character in the data item specified by the identifier is examined in turn.

If a data item referred to by the EXAMINE statement is numeric, it must consist of numeric characters and may include an operational sign. Examination starts at the leftmost character and proceeds to the right. Each character is examined in turn. When the letter 'S' is used in the PICTURE character-string of the data item description to indicate the presence of an operational sign, the sign is ignored by the EXAMINE statement.

The item being examined must have been described with USAGE DISPLAY (explicitly or implicitly). Literal-1 and literal-2 must be one-character literals; the actual characters must be consistent with the category of identifier. If the identifier is numeric, the literals must be single-digit integers (written without quotation marks). If the identifier is nonnumeric, the literals must be single characters, enclosed in quotation marks, consistent with the category of identifier. Alternatively, figurative constants may be specified, but the ALL literal option must not be used.

The TALLYING phrase of the EXAMINE statement is used for arithmetic manipulation. The description of the TALLYING function and a related example is given in Section XI, Arithmetic Computations.

## SECTION XI

### ARITHMETIC COMPUTATIONS

#### METHODS OF COMPUTATION

The manipulation and/or computation of numeric operands may be specified in COBOL:

1. In formulas within conditions.
2. With the arithmetic statements (ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT).
3. With the TALLYING phrase of the EXAMINE statement.
4. With the VARYING phrase of the PERFORM statement.
5. With the SUM counter (see Section VIII, Report Writer).

The following general rules apply to all computations:

1. Any literals specified must be numeric literals.
2. Operands must be only elementary numeric data items.
3. The sizes and formats of operands may vary within a statement. The compiler supplies object program coding for appropriate decimal point alignment and any required format conversions.
4. Editing symbols must not be specified in any operand, except in an item that receives the calculated result but is not used in the computation itself. (The exception to this rule is described under the Arithmetic Statements paragraph.)
5. The maximum size of any operand is 18 decimal digits.
6. In the Extended Instruction Set (EIS) mode, an arithmetic reference to nonnumeric data will cause an ILLEGAL EIS DATA abort message.

For arithmetic statements, the calculated result may have more integral or fractional digits than those provided by the description of the resultant data item. Therefore, the arithmetic verbs offer the SIZE ERROR option, which is used when excess high-order digits are expected, and the ROUNDED option, which is used when excess low-order digits are expected.

## FORMULAS

A formula is an algebraic expression consisting of identifiers and/or literals and arithmetic operators. Data items mentioned in formulas must be numeric elementary items.

A formula is evaluated from left to right. As in algebra, exponentiation has priority over multiplication and division, and addition and subtraction have the lowest priority. The arithmetic operators are expressed by symbols, which are shown in the order of decreasing priority in the following list:

| <u>Operation</u> | <u>Symbol</u> | <u>English Equivalent</u>            |
|------------------|---------------|--------------------------------------|
| Unary +          | +             | (Multiplication by +1)               |
| Unary -          | -             | (Multiplication by -1)               |
| Exponentiation   | **            | <u>EXPONENTIATED BY</u>              |
| Multiplication   | *             | <u>MULTIPLIED BY</u> or <u>TIMES</u> |
| Division         | /             | <u>DIVIDED BY</u>                    |
| Addition         | +             | <u>PLUS</u>                          |
| Subtraction      | -             | <u>MINUS</u>                         |

A formula must never begin with \* or / or \*\* (or their English equivalents), nor end with any arithmetic operator. Each symbol must be preceded and followed immediately by one or more spaces.

Parentheses may be used to override the assumed priority order of operations. The operations within a set of parentheses are performed, producing a single number, before operations outside the parentheses are performed. An expression is evaluated from the innermost to the outermost set of parentheses. There must be exactly as many right parentheses as left parentheses. The left parenthesis may not be immediately followed by a blank and the right parenthesis may not be preceded by a blank space.

Unless parentheses intervene, operations at a given priority level are executed in the same order as they are written, from left to right. Certain sequences of operations are considered ambiguous in algebra, but the rule just stated makes them permissible in COBOL. The following examples illustrate this convention:

| <u>Expression</u> | <u>Algebra</u>                 | <u>COBOL</u>  |
|-------------------|--------------------------------|---------------|
| A / B * C         | (A / B) * C or A / (B * C)     | (A / B) * C   |
| A / B / C         | (A / B) / C or A / (B / C)     | (A / B) / C   |
| A ** B ** C       | (A ** B) ** C or A ** (B ** C) | (A ** B) ** C |

Although such expressions are permitted in COBOL, parentheses should be used instead, as in the last column, to make the source program clearer.

The following example is unambiguous without parentheses, and illustrates the normal precedence of operations:

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

is interpreted as if it were written

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

The following usages of exponentiation are not allowed and may produce unpredictable results:

- a. The value zero exponentiated by the value zero.
- b. The value zero exponentiated by a negative value.
- c. A negative value exponentiated by a nonintegral value.

If a data item to be exponentiated can assume a negative value, a test (IF...NEGATIVE) should be arranged to bypass the exponentiation in case the value is negative.

### Unary Operators

The symbols '+' and '-' (or their English equivalents) may be used to specify 'unary' operations, and as such appear as the first symbol of a formula or following the symbols (, \*\*, \*, or /. Unary plus causes no action, but unary minus causes the negated value of the following operand to be used. (If a negative value is negated, it becomes positive.) When the plus and minus symbols are used as unary operators, they take precedence over all other operators.

### Symbol Pairs

The methods in which symbol pairs may be combined are summarized in the following table. The letter 'P' indicates a permissible pair of symbols, the character '-' represents an invalid pair, and the term 'variable' represents an identifier or literal. In this context, symbols include data-names, numeric literals, parentheses, and arithmetic operators.

## Formation of Symbol Pairs

| First Symbol | Second Symbol  |            |                 |   |   |
|--------------|----------------|------------|-----------------|---|---|
|              | Variable       | *,/,**,+,- | Unary<br>+ or - | ( | ) |
| Variable     | -              | P          | -               | - | P |
| *,/,**,+,-   | P              | -          | P               | P | - |
| Unary + or - | P <sup>1</sup> | -          | -               | P | - |
| (            | P              | -          | P               | P | - |
| )            | -              | P          | -               | - | P |

### COMMON OPTIONS IN STATEMENT FORMATS

In the statement formats of the Procedure Division, several options appear frequently; the `ROUNDED` option, the `SIZE ERROR` option, and the `CORRESPONDING` option.

The term resultant-identifier referenced in the following paragraphs is defined as that identifier associated with a result of an arithmetic operation.

#### ROUNDED Option

After decimal point alignment, the significant digits in a calculated result may extend to the right of the last digit in the resultant item's description. (In COBOL, as in arithmetic, this situation is not regarded as an error condition.) If the `ROUNDED` option is specified, the value stored in the resultant data item is determined as follows:

1. The excess digits on the right are dropped.
2. If the most significant digit of the excess digits exceeds four, the absolute magnitude of the retained value is increased by one in the least significant retained digit.
3. If the most significant digit of the excess digits does not exceed four, the retained value is unchanged.

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 rightmost integer position for which storage is allocated.

---

<sup>1</sup> Permissible only if the variable is not a literal.

The following shows the effect of specifying the ROUNDED option:

| <u>Result of<br/>Arithmetic<br/>Operation</u> | <u>PICTURE of<br/>Resultant-<br/>Identifier</u> | <u>Value Stored<br/>in Resultant-<br/>Identifier</u> |
|-----------------------------------------------|-------------------------------------------------|------------------------------------------------------|
| 3.14                                          | S9V9                                            | 3.1                                                  |
| 3.15                                          | S9V9                                            | 3.2                                                  |
| -3.14                                         | S9V9                                            | -3.1                                                 |
| -3.15                                         | S9V9                                            | -3.2                                                 |

### SIZE ERROR Option

If, after decimal point alignment, the value of the result exceeds the largest value that can be contained in the 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.

1. If the SIZE ERROR option is not specified and a size error condition occurs, the value of those resultant-identifier(s) affected is undefined. Values of resultant-identifier(s) for which no size error condition occurs are unaffected by size errors that occur for other resultant-identifier(s) during execution of this operation.
2. If the SIZE ERROR option is specified and a size error condition occurs, then the prior values of resultant-identifier(s) affected by the size errors are not altered. Values of resultant-identifier(s) for which no size error condition occurs are unaffected by size errors that occur for other resultant-identifier(s) during execution of this operation. After completion of the execution of this operation, the imperative-statement in the SIZE ERROR option is executed.

For ADD and SUBTRACT CORRESPONDING, if any of the individual operations produce a size error condition, the imperative-statement in the SIZE ERROR phrase is not executed until all of the individual additions or subtractions are completed.

### CORRESPONDING Option

In this discussion, d1 and d2 represent identifiers that refer to group items. A pair of data items, one from d1 and one from d2, correspond if the following conditions exist:

1. A data item in d1 and a data item in d2 have the same name and the same qualification up to, but not including, d1 and d2.
2. At least one of the data items is an elementary data item in the case of a MOVE statement with the CORRESPONDING option; or both of the data items are elementary numeric data items in the case of the ADD or SUBTRACT statements with the CORRESPONDING option.



3. Neither d1 nor d2 may be data items with level-number 66, 77, or 88 or be described with the USAGE IS INDEX clause.
4. A data item that is subordinate to d1 or d2 and contains a REDEFINES, RENAMES, OCCURS, or USAGE IS INDEX clause is ignored, as well as those data items subordinate to the data item that contains the REDEFINES, RENAMES, OCCURS, or USAGE IS INDEX clause. However, d1 and d2 may have REDEFINES or OCCURS clauses or be subordinate to data items with REDEFINES or OCCURS clauses.

The COBOL compiler may abort when the ADD/SUBTRACT CORRESPONDING statement is used in a program with a very large number of Data Division entries or with very long record descriptions in which 'correspondences' would exist. This abort may be avoided in subsequent compilations by increasing the memory limits. If the memory limits cannot be increased, the abort can be circumvented by replacing the ADD/SUBTRACT CORRESPONDING statement with elementary ADD/SUBTRACT statements to accomplish the desired functions.

#### ARITHMETIC STATEMENTS

The arithmetic statements are the ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT statements. They have several common rules:

1. All literals used in arithmetic statements must be numeric literals.
2. The data description of each identifier used as an operand must be that of an elementary numeric item.
3. The data descriptions of the operands need not be the same; any necessary conversion, format transformation, and/or decimal point alignment is supplied throughout the calculation.
4. The maximum size of each operand is 18 decimal digits. The composite of operands (a hypothetical data item resulting from the superimposition of specified operands in a given statement, aligned on their decimal points) must not contain more than 18 digits. The compiler ensures that enough places are carried so that significant digits are not lost during the calculation of intermediate results.
5. Editing symbols must not be specified in any operand, except in an item that receives the calculated result but is not used in the computation itself.

The resultant item of a COMPUTE statement may be an edited item. The resultant item of an ADD, DIVIDE, MULTIPLY, or SUBTRACT statement may be an edited item only when the GIVING option is specified. Operands in a computation must not be edited items in any other circumstances.

6. When the number of decimal places in a result is greater than the number of decimal places associated with the resultant-identifier, truncation occurs. However, when the ROUNDED option is specified for a resultant-identifier, the least significant digit of the resultant-identifier is increased by 1 when the most significant digit of the truncated excess is equal to or greater than 5.
7. A size error occurs when the magnitude of the calculated result exceeds the largest magnitude that can be contained in the resultant-identifier. When a size error occurs and the SIZE ERROR option is specified, the value of the resultant-identifier is not altered and the imperative-statement is executed.

## ADD Statement

The ADD statement is used to sum two or more numeric operands and store the result.

When Format 1 or Format 2 of the ADD statement is used, each identifier must refer to an elementary numeric item, except that in Format 2 the identifier following the word GIVING must refer either to an elementary numeric item or to an elementary numeric edited item.

When Format 1 of the ADD statement is used, the values of the operands preceding the word TO are added together. That sum is then added to the current value of each identifier-m, identifier-n, ..., and the result is stored in each resultant-identifier: identifier-m, identifier-n, ..., respectively.

When Format 2 of the ADD statement is specified, the values of the operands preceding the word GIVING are added together. That sum is then stored as the new value of each identifier-m, identifier-n, ..., which are the resultant-identifiers.

When Format 3 of the ADD statement is used, the data items in identifier-1 are added to and stored in corresponding data items in identifier-2.

### Examples:

```
ADD 0.5, RATE OF PAY-FILE GIVING TOTAL.
```

```
ADD TOTAL-RECVE, TO ON-HAND-QTY ROUNDED.
```

```
ADD VALUE-1 OF FILE-A, VALUE-2 OF FILE-B GIVING VALUE-3 OF FILE-C;
ON SIZE ERROR GO TO ERROR-RTN.
```

```
ADD CORRESPONDING ABLE TO BAKER (I) ROUNDED.
```

## COMPUTE Statement

The COMPUTE statement allows the user to combine arithmetic operations without the restrictions on composite of operands and/or receiving data items imposed by the arithmetic statements ADD, DIVIDE, MULTIPLY, or SUBTRACT.

The execution of a COMPUTE statement causes identifier-1 to receive the value of identifier-2, literal-1, or the computed value of the formula, whichever is specified.

The reserved words FROM and EQUALS and the symbol '=' are synonymous in the COMPUTE statement; they may be used interchangeably.

If the SIZE ERROR phrase is not specified, the final results will not be truncated when the receiving identifier is described as COMPUTATIONAL or COMPUTATIONAL-1 unless more than one receiving identifier is specified in the COMPUTE statement or span overflow has occurred.

Editing symbols must not be specified in any operand, except in an item that receives the calculated result but is not used in the computation itself.

The resultant item of a COMPUTE statement may be an edited item. The resultant item of an ADD, DIVIDE, MULTIPLY, or SUBTRACT statement may be an edited item only when the GIVING option is specified. Operands in a computation must not be edited items in any other circumstances.

**Examples:**

```
COMPUTE GROSS-PAY OF PAY-FILE ROUNDED = HRS-WORKED * RATE -
WEEKLY-TAX.

COMPUTE QTY-ON-HAND = OLD-QTY + NO-RECVD - QTY-SHIPED.

COMPUTE AVG-INCREASE = (END-PAY - START-PAY) / END-PAY.

COMPUTE YTD-FICA = YTD-FICA + (GROSS-PAY * 0.06).
```

**DIVIDE Statement**

The DIVIDE statement is used to divide one numeric data item into another and to set the value of a data item equal to the result.

When Format 1 of the DIVIDE statement is used, the value of identifier-1 or literal-1 is divided into the value of identifier-2. The value of the dividend (identifier-2) is replaced by this quotient; similarly for identifier-1 or literal-1 and identifier-3, etc.

When Format 2 is used, the value of identifier-1 or literal-1 is divided into identifier-2 or literal-2 and the result is stored in identifier-3, identifier-4, etc.

When Format 3 is used, the value of identifier-1 or literal-1 is divided by the value of identifier-2 or literal-2 and the result is stored in identifier-3, identifier-4, etc.

Formats 4 and 5 are used when a remainder from the division operation is desired, namely identifier-4. The remainder in COBOL is defined as the result of subtracting the product of the quotient (identifier-3) and the divisor from the dividend. If ROUNDED is used, the quotient used to calculate the remainder is an intermediate field that contains the quotient of the DIVIDE statement, truncated rather than rounded.

In a DIVIDE statement, the COBOL rules specified for calculating the REMAINDER generally preclude the GIVING field (identifier-3) from also being designated as either the divisor or the dividend.

**Example:**

```
DIVIDE B INTO A GIVING B REMAINDER X.
```

The above coding should not be used unless the following conditions exist:

1. The values contained in all of the identifiers are single-precision values.
2. The identifiers are all described with USAGE COMPUTATIONAL-1.

When the SIZE ERROR phrase is used in Formats 4 and 5, the following rules apply:

1. If the size error occurs on the quotient, the contents of identifier-3 will not be changed but the result of the divide will be used in computing the remainder.
2. If the size error occurs on the remainder, the contents of the data item referenced by identifier-4 will remain unchanged. However, as with other instances of multiple results of arithmetic statements, analysis must be performed to determine which situation has occurred.

A nonstandard statement of the form 'DIVIDE identifier-1 BY identifier-2' will cause the object code compiled to replace the starting value of the dividend (identifier-1) with the computed value of the quotient obtained from the division. This nonstandard application is not recommended.

Examples:

```
DIVIDE TOTAL BY NUMBER GIVING AVERAGE.
```

```
DIVIDE 100.00 INTO K2H GIVING VALUE-1 OF FILE-16 ROUNDED.
```

```
DIVIDE A26 INTO R17K.
```

#### MULTIPLY Statement

The MULTIPLY statement is used to multiply numeric data items and to set the values of data items equal to the results.

When Format 1 of the MULTIPLY statement is used, the value of identifier-1 or literal-1 is multiplied by the value of identifier-2. The value of identifier-2 is then replaced by the product. Similar action occurs for identifier-1 or literal-1 and identifier-3, etc.

When Format 2 is used, the value of identifier-1 or literal-1 is multiplied by the value of identifier-2 or literal-2 and the product is stored in identifier-2, identifier-4, etc.

Examples:

```
MULTIPLY 0.18 BY PAY GIVING TAX, ON SIZE ERROR GO TO ERROR-RTN.
```

```
MULTIPLY A OF FILE-1 BY B OF FILE-2 GIVING C OF FILE-3.
```

```
MULTIPLY 3.1416 BY R1.
```

## SUBTRACT Statement

The SUBTRACT statement is used to subtract one, or the sum of two or more, numeric data items from one or more items, and set the values of one or more items equal to the results.

When Format 1 of the SUBTRACT statement is used, all literals or identifiers preceding the word FROM are added together and this total is subtracted from identifier-m, identifier-n, etc., and the differences are stored as the new values of identifier-m, identifier-n, etc.

When Format 2 is used, 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, identifier-o, etc.

When Format 3 is used, data items in identifier-1 are subtracted from and stored into corresponding data items in identifier-2.

### Examples:

```
SUBTRACT UNION-DUES OF MASTR-PAYROL FROM ADJUSTED-PAY OF
MASTR-PAYROL.
```

```
SUBTRACT RECEIPTS OF TRANSAC-FILE FROM ON-ORDER-QTY OF
ORDER-FILE GIVING ADJ-ORDR-QTY, SIZE ERROR GO TO ZERO-RTN.
```

```
SUBTRACT A FROM B.
```

```
SUBTRACT CORRESPONDING ABLE FROM BAKER (I) ROUNDED.
```

### Multiple Results in Arithmetic Statements

The ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT statements may have multiple results. Such statements behave as though they had been written as:

1. Statements that perform all arithmetic necessary to arrive at the result to be stored in the receiving items, and store that result in a temporary memory location.
2. A sequence of statements transferring or combining the value of this temporary location with a single result. These statements are considered to be written in the same left-to-right sequence in which the multiple results are listed.

The result of the statement

```
ADD a, b, c, TO c, d (c), e
```

is equivalent to

```
ADD a, b, c, GIVING temp
ADD temp TO c
ADD temp TO d (c)
ADD temp TO e
```

where 'temp' is an intermediate result item provided by the compiler. The data item d is subscripted to show that the value of c is determined prior to its use as a subscript.

## Overlapping Operands

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

## Precision in Arithmetic Calculations

The precision obtained in arithmetic calculations varies among manufacturers. COBOL uses the COMPUTE statement to perform complex arithmetic calculations which enhance precision. In a series of arithmetic operations within a single COMPUTE statement, the precision of intermediate results determines the precision of the results of each subsequent arithmetic operation. Therefore, the compiler, in evaluating a complex arithmetic statement, maintains decimal precision for all intermediate results. Truncation is carried out only after the move to the final resultant field. This means that in a COMPUTE statement, wherein all data items are described as integer values, a fractional intermediate result may be obtained from a divide operation. That fractional intermediate result is carried intact into subsequent arithmetic operations. The fractional places are truncated only upon the move to the final resultant field.

Example:

```
COMPUTE A = B - ((C + D) / 2).
```

where

```
A = PIC 99.
B = PIC 99 VALUE 9.
C = PIC 99 VALUE 6.
D = PIC 99 VALUE 1.
```

The result of the above calculation would be:

1.  $\frac{6+1}{2} = 3.5$
2.  $9 - 3.5 = 5.5$
3. MOVE 5.5 to A truncates the .5 giving the final result 5.

The maintenance of decimal precision in COMPUTE statements also affects the results that are obtained with the ROUNDED phrase. The rounding operation is accomplished on the move to the final resultant field just before truncation. Thus, in the previous example

COMPUTE A ROUNDED = B - ((C + D) / 2).

the sequence of operations would be:

1.  $\frac{6+1}{2} = 3.5$
2.  $9 - 3.5 = 5.5$
3. MOVE to A rounds 5.5 to 6.0; then the .0 is truncated giving a final result of 6.

When data items are described with USAGE COMPUTATIONAL or COMPUTATIONAL-2, ultimate accuracy for maximum length composite of operands (18 digits) in arithmetic statements may not be attainable due to floating-point hardware limitations.

#### TALLYING PHRASE (EXAMINE STATEMENT)

The EXAMINE statement is used to replace and/or count the number of occurrences of a given character in a data item. Two phrases, the REPLACING phrase and the TALLYING phrase, may be specified in the statement. When Format 2, the EXAMINE...REPLACING statement is used, it is considered a data movement procedure and, as such, is described in Section X.

The EXAMINE...TALLYING statement may be used for either or both of two basic purposes:

1. To scan a data item, counting the number of occurrences of a given character (literal-1).
2. To modify the value of an item, by replacing certain characters in the original value with a new character (literal-2).

The TALLYING phrase causes an integral count to be placed in TALLY (a special register). The significance of the count depends upon the options specified:

| <u>Option</u> | <u>TALLY Value Represents</u>                                                                                                |
|---------------|------------------------------------------------------------------------------------------------------------------------------|
| ALL           | The number of occurrences of literal-1 throughout the item.                                                                  |
| LEADING       | The number of occurrences of literal-1 prior to encountering a character other than literal-1.                               |
| UNTIL FIRST   | The number of occurrences of other characters not equal to literal-1 encountered prior to the first occurrence of literal-1. |

When both the TALLYING and REPLACING phrases are specified in Format 1, each character tallied according to the rules given above is also replaced with literal-2.

For nonnumeric data items, examination starts at the leftmost character and proceeds to the right. Each character in the data item specified by the identifier is examined in turn.

If a data item referred to by the EXAMINE statement is numeric, it must consist of numeric characters and may possess an operational sign. Examination starts at the leftmost character and proceeds to the right. Each character is examined in turn. When the letter 'S' is used in the PICTURE character-string of the data item description to indicate the presence of an operational sign, the sign is ignored by the EXAMINE statement.

The item being examined must have been described with USAGE DISPLAY (explicitly or implicitly). Literal-1 and literal-2 must be one-character literals; the actual characters must be consistent with the category of identifier. If the identifier is numeric, the literals must be single-digit integers (written without quotation marks). If the identifier is nonnumeric, the literals must be single characters, enclosed in quotation marks, consistent with the category of identifier. Alternatively, figurative constants may be specified, but the ALL literal option must not be used.

#### EXAMPLE OF EXAMINE...TALLYING

An example of a basic EXAMINE statement that illustrates an application of the tallying function is presented below:

```
.
.
WORKING-STORAGE SECTION.
01 EXAMINE-DATA.
 02 E-1 PICTURE IS A(10) VALUE IS "ABCABCBCD".
 02 E-2 PICTURE IS 9(10) VALUE IS 0150250350.
.
.
PROCEDURE DIVISION.
EXAMINE-TEST-1.
 EXAMINE E-1 TALLYING UNTIL FIRST "D".
 IF TALLY EQUAL TO 9 PERFORM PASS, ELSE GO TO FAIL.
.
.
EXAMINE-TEST-2.
 EXAMINE E-2 TALLYING ALL 5 REPLACING BY 0.
 IF TALLY EQUAL TO 3 NEXT SENTENCE, ELSE GO TO FAIL.
 IF E-2 EQUAL TO 0100200300 PERFORM PASS, ELSE GO TO FAIL.
.
.
```



VARYING PHRASE (PERFORM STATEMENT)

The VARYING phrase of the PERFORM statement allows the referenced numeric operand to be manipulated during the execution of the PERFORM statement in such a manner that its value remains in a changed state following the execution of the PERFORM statement.

For a detailed description of the manner in which this VARYING manipulation takes place, refer to the PERFORM statement in the COBOL Reference Manual.

## SECTION XII

### CONDITIONAL PROCEDURES

#### CONDITIONS

A condition enables the object program to select between alternate paths of control, depending upon the truth value of a test. A condition is one of the following:

- a. Relation condition
- b. Sign condition
- c. Class condition
- d. Condition-name condition
- e. Switch-status condition
- f. NOT condition
- g. condition  $\left\{ \begin{array}{c} \text{AND} \\ \text{OR} \end{array} \right\}$  condition  $\left[ \left\{ \begin{array}{c} \text{AND} \\ \text{OR} \end{array} \right\} \text{condition} \right] \dots$

Any condition may be enclosed in parentheses. The truth value of a parenthesized condition is determined from the evaluation of the truth values of its constituents. A parenthesized condition is a condition in the sense of the last two items of the preceding list.

#### Simple Conditions

There are five types of simple condition tests. These tests and some of the acceptable formats for stating them are described below.

#### RELATION CONDITION

A relation condition involves a comparison of two operands, each of which may be the data item referenced by an identifier, a literal, or the value resulting from an arithmetic-expression. The comparison of two literals is not permitted. Comparison of numeric operands is permitted regardless of their individual usages. All other comparisons require that the USAGE of the operands being compared is the same. If either of the operands is a group item, the nonnumeric comparison rules apply.

## Comparison of Numeric Operands

For numeric operands, a comparison results in the determination that the algebraic value of one of the operands is less than, equal to, or greater than the other. The operand length, in terms of the number of digits, is not significant. Zero is considered to represent a unique value regardless of the length, sign, or implied decimal point location.

Comparison of these operands is permitted regardless of the manner in which their usage is described. Unsigned numeric operands are considered to be positive for comparison purposes.

## Comparison of Nonnumeric Operands

For two nonnumeric operands, or one numeric (excluding the operational sign) and one nonnumeric operand, a comparison results in the determination that one of the operands is less than, equal to, or greater than the other with respect to an ordered character set. If one of the operands is specified as numeric, it must be an integer data item or an integer literal. Numeric and nonnumeric operands may be compared only when their usage is the same, implicitly or explicitly. Except when USAGE of the operands is DISPLAY-2, the character set order is determined by the standard collating sequence. For DISPLAY-2 operands, the order is determined by the commercial collating sequence. The collating sequences are listed in Appendix D.

If the operands are of equal size, characters in corresponding character positions are compared starting from the high-order end and continuing until either a pair of unequal characters is encountered or the low-order end of the item is reached, whichever comes first. The items are determined to be equal when the low-order end is reached.

The first encountered pair of unequal characters is compared for relative location in the collating sequence. The operand containing that character which is positioned higher in the collating sequence is determined 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 sufficient spaces to make the operands of equal size. If this process exhausts the characters of the operand of lesser size, then the operand of lesser size is less than the operand of larger size unless the remainder of the operand of larger size consists solely of spaces, in which case the two operands are equal.

## Comparisons Involving Index-Names and/or Index Data Items

Full relation tests may be made between:

- a. Two index-names. The result is the same as if the corresponding occurrence numbers are compared.
- b. An index-name and a data item (other than an index data item) or a literal. The occurrence number that corresponds to the value of the index-name is compared to the data item or literal.

- c. 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 or literal not specified in a, b, or c above is undefined.

#### SIGN CONDITION

The sign condition determines whether or not the algebraic value of an elementary numeric data item or an arithmetic-expression is less than, equal to, or greater than zero. (The word IF is not part of the condition but is included in the formats to improve readability.) The general format for a sign condition is:

$$\underline{\text{IF}} \left\{ \begin{array}{l} \text{identifier} \\ \text{arithmetic-expression} \end{array} \right\} \text{ IS } [ \text{NOT} ] \left\{ \begin{array}{l} \text{POSITIVE} \\ \text{NEGATIVE} \\ \text{ZERO} \end{array} \right\}$$

An operand is POSITIVE only if its value is greater than zero, NEGATIVE if its value is less than zero, and ZERO if its value is equal to zero. An operand whose value is zero is NOT POSITIVE and an operand whose value is zero is NOT NEGATIVE; the value zero is considered neither positive nor negative.

#### CLASS CONDITION

The class of any item can be tested as follows:

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

The usage of identifier must be explicitly or implicitly DISPLAY. The ALPHABETIC test cannot be used with an item whose data description describes the item as numeric. The item being tested is determined to be alphabetic only if the contents consist of the alphabetic characters 'A' through 'Z' and the space.

The NUMERIC test cannot be used with an item whose data description describes the item as alphabetic or as a group item composed of elementary items whose data description indicates the presence of operational sign(s). If the data description of the item being tested does not indicate the presence of an operational sign, the item being tested is determined to be numeric only if the contents are numeric and an operational sign is not present.

The following are examples of class tests:

```
77 A PIC X(18) VALUE "00ABCD".
77 B REDEFINES A PIC S9(18).

77 C PIC X(2) VALUE "22".

77 D PIC S9 VALUE -1.
77 E REDEFINES D PIC 9.
```

|    |   |         | Condition |
|----|---|---------|-----------|
| IF | A | NUMERIC | False     |
| IF | B | NUMERIC | False     |
| IF | C | NUMERIC | True      |
| IF | E | NUMERIC | False     |

#### CONDITION-NAME CONDITION

In a condition-name condition, a conditional variable is tested to determine whether or not its value is equal to one of the values associated with a condition-name in the Data Division. The general format for the condition-name condition is:

```
IF [NOT] condition-name
```

If the condition-name is associated with a range or ranges of values (that is, the VALUES ARE clause contains at least one 'literal THRU literal' phrase), then the conditional variable is tested to determine whether or not its value falls in this range, including the end values.

The rules for comparing a conditional variable with a condition-name value are the same as those specified for relation conditions.

The result of the test is true if one of the values corresponding to the condition-name equals the value of its associated conditional variable.

#### Example:

Let MARRIED, SINGLE, WIDOW, WIDOWER, DIVORCED correspond to the actual values 1, 2, 3, 4, 5, respectively, of a field called MARITAL-STATUS. The conditional statement

```
IF SINGLE. . .
```

would generate in the object program a test of the value of the conditional variable MARITAL-STATUS against the value '2'.

## SWITCH-STATUS CONDITION

In the SPECIAL-NAMES paragraph of the Environment Division, a condition-name may be associated with the ON or OFF status of a software switch. The switch is ON when its value is one (1) and OFF when its value is zero (0). The status of such a switch may then be tested with a statement having the IF NOT condition-name format. The results of this test are determined using the parameters contained in the table in the Switch-Status Condition paragraph in the COBOL Reference Manual.

Additional information is contained in the Switches paragraph in Section VI.

### Compound Conditions

Simple conditions can be combined with logical operators according to specified rules to form compound conditions. The logical operators AND, OR, and NOT must be preceded by a space and followed by a space. The meaning of the logical operators follows:

OR Logical Inclusive Or

AND Logical Conjunction

NOT Logical Negation

The general format of a compound condition is:

$$\underline{\text{IF}} \text{ condition-1 } \left\{ \begin{array}{c} \underline{\text{AND}} \\ \underline{\text{OR}} \end{array} \right\} \text{ condition-2 } \left[ \left[ \begin{array}{c} \underline{\text{AND}} \\ \underline{\text{OR}} \end{array} \right] \text{ condition-n } \right] \dots$$

The word IF is shown to improve readability. Each condition can be either a relation condition, a sign condition, a class condition, a condition-name condition, or a switch-status condition.

The following are examples of compound conditions:

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

AGE IS GREATER THAN 25 OR MARRIED

STOCK-ON-HAND IS LESS THAN DEMAND OR STK-SUPPLY IS GREATER THAN DEMAND PLUS INVENTORY

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

STK-ACCT IS GREATER THAN 72 AND (STK-NUMBER IS LESS THAN 100 OR STK-NUMBER EQUALS 76290)

Letting A and B represent simple conditions, the following table defines the interpretation of AND, OR, and NOT in compound conditions:

| Condition |       | Condition and Value |         |        |
|-----------|-------|---------------------|---------|--------|
| A         | B     | NOT A               | A AND B | A OR B |
| True      | True  | False               | True    | True   |
| False     | True  | True                | False   | True   |
| True      | False | False               | False   | True   |
| False     | False | True                | False   | False  |

Thus, if A is true and B is false, the expression A AND B is false, while the expression A OR B is true.

The following table indicates the methods in which conditions and logical operators may be combined:

| FIRST SYMBOL | SECOND SYMBOL  |    |     |     |   |   |
|--------------|----------------|----|-----|-----|---|---|
|              | Condition      | OR | AND | NOT | ( | ) |
| Condition    | -              | P  | P   | -   | - | P |
| OR           | P              | -  | -   | P   | P | - |
| AND          | P              | -  | -   | P   | P | - |
| NOT          | P <sup>1</sup> | -  | -   | -   | P | - |
| (            | P              | -  | -   | P   | P | - |
| )            | -              | P  | P   | -   | - | P |

'P' indicates that the pair is permissible and '-' indicates that the pair is not permissible. Thus, the pair 'OR NOT' is permissible, while the pair 'NOT OR' is not permissible.

<sup>1</sup>Permissible only if the condition itself does not contain a NOT.

The rules for determining the logical value (true or false) of a compound condition are as follows:

1. If AND is the only logical connective used, then the compound condition is true if each of the simple conditions is true.
2. If OR is the only logical connective used, then the compound condition is true if one or more of the simple conditions is true.
3. If both AND and OR appear, then there are two cases to consider, depending on whether or not parentheses are used.
  - a. Parentheses can be used to indicate grouping. They must always be paired, as in algebra, and the expressions within the parentheses will be evaluated first. The precedence of nested parenthetical expressions is the same as in algebra. That is, the innermost parenthetical expressions are evaluated first.
  - b. If parentheses are not used, then the conditions are grouped first according to AND, proceeding from left to right, and then by OR, proceeding from left to right. The logical operator AND has precedence over the logical operator OR in the same sense that the arithmetic operator \* (multiplication) has precedence over the arithmetic operator + (addition).
4. When NOT precedes a parenthesized condition, it reverses the logical value of the parenthesized condition; that is, NOT (condition) is true when (condition) is false. For example, NOT (A AND B) is true if either A or B is false. Thus, NOT (A AND B) is equivalent to NOT A OR NOT B, and is true when A and B are not both true (i.e., when either is false or both are false). Similarly, NOT (A OR B) is equivalent to NOT A AND NOT B, and is true only when A and B are both false.

Examples:

1. To evaluate  $C1 \text{ AND } (C2 \text{ OR NOT } (C3 \text{ OR } C4))$ , apply rule 3a (above) and successively reduce this by substituting as follows:  
$$\text{Let } C5 \text{ equal "C3 OR C4" resulting in } C1 \text{ AND } (C2 \text{ OR NOT } C5)$$
$$\text{Let } C6 \text{ equal "C2 OR NOT } C5" \text{ resulting in } C1 \text{ AND } C6$$

This can be evaluated using the preceding Condition/Condition and Value table.
2. To evaluate  $C1 \text{ OR } C2 \text{ AND } C3$ , apply rule 3b (above) and reduce this to  $C1 \text{ OR } (C2 \text{ AND } C3)$ , which can then be reduced as in Example 1.
3. To evaluate  $C1 \text{ AND } C2 \text{ OR NOT } C3 \text{ AND } C4$ , group first by AND from left to right, resulting in:  
$$(C1 \text{ AND } C2) \text{ OR } (\text{NOT } C3 \text{ AND } C4)$$

which can now be evaluated as in the first example.
4. To evaluate  $C1 \text{ AND } C2 \text{ AND } C3 \text{ OR } C4 \text{ OR } C5 \text{ AND } C6 \text{ AND } C7 \text{ OR } C8$ , group from the left by AND to produce:  
$$((C1 \text{ AND } C2) \text{ AND } C3) \text{ OR } C4 \text{ OR } ((C5 \text{ AND } C6) \text{ AND } C7) \text{ OR } C8$$

which can now be evaluated as in Example 1.



## Abbreviated Combined Relation Conditions

Only conditions involving full relation tests have three terms (a subject, a relation, and an object). To simplify writing lengthy expressions, COBOL allows the omission of some of these terms in certain forms of compound conditions.

In the following general format,

operand-1 represents  $\left\{ \begin{array}{l} \text{identifier-1} \\ \text{literal-1} \\ \text{formula-1} \end{array} \right\} .$

The general format of a compound condition containing only full relation tests is as follows:

IF operand-1 (relation-1) operand-2  $\left\{ \begin{array}{l} \text{AND} \\ \text{OR} \end{array} \right\}$  operand-3  
  
(relation-2) operand-4  $\left\{ \begin{array}{l} \text{AND} \\ \text{OR} \end{array} \right\}$  ...operand-n1  
  
(relation-n) operand-n2

Of the five examples given previously under the Compound Conditions paragraph, only the first, third, and fifth examples are of this form.

### ABBREVIATION 1

When a relational compound condition has the same term immediately preceding each relation, only the first term must be written. Thus:

IF operand-1 (relation-1) operand-2  $\left\{ \begin{array}{l} \text{AND} \\ \text{OR} \end{array} \right\}$  (relation-2)  
  
operand-3  $\left\{ \begin{array}{l} \text{AND} \\ \text{OR} \end{array} \right\}$  (relation-3) ...  $\left\{ \begin{array}{l} \text{AND} \\ \text{OR} \end{array} \right\}$  (operand-n)

is interpreted as if operand-1 had appeared immediately preceding each relation-n.

This form of abbreviation is applicable regardless of the presence or absence of parentheses.

Referring back to the first example, the example could also be written as:

IF AGE IS LESS THAN MAX-AGE AND GREATER THAN 20

As another illustration of this abbreviation, note that

IF A EQUALS B OR EQUALS C AND IS GREATER THAN D

is an abbreviation for

IF A EQUALS B OR A EQUALS C AND A IS GREATER THAN D

which is equivalent to:

IF A EQUALS B OR (A EQUALS C AND A IS GREATER THAN D).

#### ABBREVIATION 2

The second type of abbreviation allowed is where both the subject and the relation are common. In this case, only the first occurrence of the subject and the relation are written. Thus:

IF operand-1 (relation) operand-2 { AND } operand-3

{ AND } ... { AND } operand-n

is equivalent to

IF operand-1 (relation) operand-2 { AND } operand-1

(relation) operand-3 { AND } ... { AND } operand-1

(relation) operand-n

This form of abbreviation is applicable regardless of the presence or absence of parentheses.

As an illustration of this form, note that the expression

IF A = B OR C AND D

is equivalent to

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

which is in turn equivalent to:

IF A = B OR (A = C AND A = D).

Using both types of abbreviations, any sequence of full relation tests can be formed into a single sentence regardless of what verbs, keywords, or other types of tests appear between them.

In summary, when relation conditions are written in a consecutive sequence, any relation condition except the first may be abbreviated by:

1. Omitting the subject of the relation condition, or
2. Omitting the subject and relational operator of the relation condition.

Within a sequence of relation conditions, both forms of abbreviations may be used. The effect of using them is as if the omitted subject were replaced by the last preceding stated subject or the omitted relational operator were replaced by the last preceding stated relational operator.

Ambiguity may result from using 'NOT' in conjunction with abbreviations. In this event, NOT is interpreted as a logical operator rather than as part of a relational operation. Thus:

a > b AND NOT > c OR d

is equivalent to:

a > b AND NOT a > c OR a > d     or

a > b AND (NOT a > c) OR a > d

The following examples show an abbreviated form followed by its equivalent expansion:

1. IF A = B AND C > D OR = B MOVE X TO Y.  
IF A = B AND C > D OR C = B MOVE X TO Y.
2. IF A = B MOVE X TO Y; ELSE IF GREATER THAN C ADD M TO N.  
IF A = B MOVE X TO Y; ELSE IF A IS GREATER THAN C ADD M TO N.
3. IF A = B OR IS GREATER THAN B MOVE C TO A; IF GREATER THAN B ADD B TO A.  
IF A = B OR A IS GREATER THAN B MOVE C TO A; IF A IS GREATER THAN B ADD B TO A.
4. IF A > B OR EQUALS B AND X EQUALS Y IF GREATER THAN B MOVE C TO D.  
IF A > B OR (A EQUALS B AND X EQUALS Y) IF X IS GREATER THAN B MOVE C TO D.
5. IF A EQUALS B AND (C OR D AND E IS GREATER THAN Y) OR LESS THAN Z MOVE M TO N.  
IF (A EQUALS B AND (A EQUALS C OR (A EQUALS D AND E IS GREATER THAN Y))) OR E IS LESS THAN Z MOVE M TO N.
6. IF A = B OR C AND D MOVE X TO Y.  
IF A = B OR A = C AND A = D MOVE X TO Y.

#### Use of the NOT Operator

Simple IF sentences may be preferred when making a conditional test to avoid the possibility of misusing the NOT logical operator and to interpret the source language more clearly. When the NOT logical operator is used in IF sentences, it must precede a left parenthesis or a simple condition which does not contain a 'NOT'. The effect of the NOT logical operator is as if the statement were interpreted by the compiler according to the following examples:

1. Relational operators, sign conditions, or class conditions are complemented; i.e., EQUAL is interpreted as NOT EQUAL, NOT EQUAL is interpreted as EQUAL:  
  
IF A = B AND NOT (C = D)  
  
is interpreted as:  
  
IF A = B AND C NOT = D.
2. The logical operator OR is interpreted as AND:  
  
IF E NOT = F AND NOT (J = 4 OR 5)  
  
is interpreted as:  
  
IF E NOT = F AND (J NOT = 4 AND J NOT = 5).
3. The logical operator AND is interpreted as OR:  
  
IF E = F AND NOT (J = 4 AND 5)  
  
is interpreted as:  
  
IF E = F AND (J NOT = 4 OR J NOT = 5).

4. The NOT preceding a conditional variable is removed:

```
IF E = F AND NOT (GREEN AND NOT RED)
```

is interpreted as:

```
IF E = F AND (NOT GREEN OR RED).
```

5. The NOT logical operator applies to the entire portion of the statement within the parentheses which it precedes. Since many logical relationship levels may be preceded by NOT, there is a net effect, at each logical level, of all previous appearances of NOT in that statement:

```
IF A = B OR NOT (B = C OR D AND NOT (E > F AND NOT (G = H)))
```

is interpreted as:

```
IF A = B OR (B NOT = C AND D OR1 (E > F AND (G NOT = H))).
```

<sup>1</sup>NOTE: The NOT at level one and the NOT at level two cancel each other, making the net effect at level two of no 'NOT'.

6. The following IF sentence illustrates a frequent misuse of the NOT = operator in relation to the OR and AND logical connectors:

```
IF C NOT = 9 OR 10 DISPLAY '9'.
```

This sentence always results in displaying '9', since the compiler logic analysis is:

```
IF C NOT = 9 DISPLAY '9' or
```

```
IF C = 9 AND C NOT = 10 DISPLAY '9'.
```

If C is not equal to 9, a branch to DISPLAY '9' occurs and no check is made against the remaining parameters. If C equals 9, it will not equal 10 and a branch to DISPLAY '9' will result.

7. To achieve the result desired in the above sentence using the NOT EQUAL operator, the sentence could read:

```
IF C NOT = 9 AND 10 DISPLAY '9'.
```

The compiler logic analysis is:

```
IF C NOT = 9 AND C NOT = 10 DISPLAY '9'.
```

In this case, if C is not equal to 9, a comparison will be made to 10.

A positive sentence to obtain the same result could read:

```
IF C = 9 OR 10 NEXT SENTENCE ELSE DISPLAY '9'.
```

8. The following sentences are additional examples of the NOT logical operator and compiler logic analyses.

| Source Language Sentences                                  | Compiler Logic Translation                                |
|------------------------------------------------------------|-----------------------------------------------------------|
| IF NOT (J = 4 OR 5), GO TO X.                              | IF J NOT = 4 AND J NOT = 5, GO TO X.                      |
| IF NOT (J = 4 AND 5), GO TO X.                             | IF J NOT = 4 OR J NOT = 5, GO TO X.                       |
| IF NOT (J NOT = 4 AND 5), GO TO X.                         | IF J = 4 OR 5, GO TO X.                                   |
| IF NOT (J NOT = 4 OR 5), GO TO X.                          | IF J = 4 AND 5, GO TO X.                                  |
| IF NOT (GREEN AND NOT RED), GO TO X.                       | IF NOT GREEN OR RED, GO TO X.                             |
| IF NOT (GREEN AND RED), GO TO X.                           | IF NOT GREEN OR NOT RED, GO TO X.                         |
| IF NOT (GREEN OR RED), GO TO X.                            | IF NOT GREEN AND NOT RED, GO TO X.                        |
| IF NOT (GREEN OR RED AND L IS POSITIVE), GO TO X.          | IF NOT GREEN AND (NOT RED OR L IS NOT POSITIVE), GO TO X. |
| IF NOT (GREEN AND NOT RED AND L IS NOT POSITIVE), GO TO X. | IF NOT GREEN OR RED OR L IS POSITIVE, GO TO X.            |
| IF NOT (GREEN OR RED AND BROWN), GO TO X.                  | IF NOT GREEN AND NOT RED OR NOT BROWN, GO TO X.           |
| IF NOT (GREEN OR (RED OR BROWN)), GO TO X.                 | IF NOT GREEN AND NOT RED AND NOT BROWN, GO TO X.          |
| IF NOT (GREEN AND RED OR BROWN), GO TO X.                  | IF (NOT GREEN OR NOT RED) AND NOT BROWN, GO TO X.         |
| IF NOT ((GREEN AND RED) OR BROWN), GO TO X.                | IF (NOT GREEN OR NOT RED) AND NOT BROWN, GO TO X.         |

9. The following positive IF sentences illustrate a frequent misuse of the AND and OR logical connectors:

| Source Language Sentences                                 | Compiler Logic Translation                                    |
|-----------------------------------------------------------|---------------------------------------------------------------|
| IF E = F AND NOT GREEN OR RED AND L IS POSITIVE, GO TO X. | IF (E = F AND NOT GREEN) OR (RED AND L IS POSITIVE), GO TO X. |
| IF E = F AND J = 4 OR 5, GO TO X.                         | IF (E = F AND J = 4) OR (J = 5) GO TO X.                      |
| IF NOT GREEN AND RED OR L IS NOT POSITIVE, GO TO X.       | IF (NOT GREEN AND RED) OR (L IS NOT POSITIVE), GO TO X.       |

## Evaluation Rules for Conditions

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

1. Arithmetic-expression
2. All relational operators
3. NOT
4. AND
5. OR

## SECTION XIII

### TABLE HANDLING

#### DESCRIPTION OF TABLE HANDLING

A table is a named set of data items arranged in some meaningful order. COBOL tables are defined by including the OCCURS clause in the data description entry.

The OCCURS clause specifies the number of times a data item is to be repeated. A data item having an OCCURS clause is called a table element, and the name and description of the table element applies to each repetition or occurrence of the data item. Each repetition of the table element is associated with an occurrence number.

When a table item is referenced, it is necessary to indicate either by subscripting or by indexing which occurrence of the table item is intended. A table item can be any of the following:

1. A table element.
2. An item within a group item which is a table element.
3. A condition-name associated with a table element.
4. A condition-name associated with an item within a group table element.

In COBOL, a table item with as many as three dimensions may be referenced using either subscripting or indexing. The general format is:

data-name (a<sub>1</sub> [ , a<sub>2</sub> [ , a<sub>3</sub> ] ] )

where: each 'a' represents an occurrence number of a dimension of the table, expressed in the form of an index-name or a subscript.

Subscripts are nonzero positive integer values expressed as numeric literals or as data-names.

An index-name is a word containing at least one alphabetic character that names an index (pointer) uniquely associated with a specific table.

One to three subscript or index levels may appear in the reference, depending upon how many dimensions are in that table of which the data-name is an element. There must be one subscript or index-name for each OCCURS clause in the defined hierarchy that contains data-name, including data-name itself. The data-name may be qualified.



Multiple subscripts and indexes are written from left to right in descending order of inclusiveness, higher to lower. A multidimensional table may be considered as a group of nested single-dimensional tables, with the outermost table the most inclusive and the innermost table the least inclusive.

An example of a three-dimensional table is shown below with representative subscript references to table elements:

```
01 TABLE.
 02 STORE OCCURS 12 TIMES.
 03 STORE-NO...
 03 DEPARTMENT OCCURS 7 TIMES.
 04 DEPT-NO...
 04 SALE OCCURS 4 TIMES.
```

```
.
.
STORE (11)
STORE-NO (12)

DEPARTMENT (10,5)
DEPT-NO (12,7)

SALE (1,1,1)
SALE (12,7,4)
```

#### SUBSCRIPTING

One method of specifying occurrence numbers is to append one, two, or three subscripts to the data-name of the table element. A subscript is an integer value that specifies the occurrence number of an element within a table.

The subscript can be represented either as a numeric literal or as a data-name that may be qualified but not itself subscripted and is defined elsewhere as an elementary numeric data item. Series 60/6000 COBOL handles data-name subscripts more efficiently if they are described with USAGE COMP-1. When subscripts are specified, it is the responsibility of the user to ensure that the subscript is within the OCCURS range for the referenced table.

The lowest valid subscript is 1. The highest valid subscript is the maximum occurrence number of the item, as specified in the OCCURS clause. An invalid subscript will cause unpredictable results at program execution.

Data-names may be used in place of numeric literal subscripts, and may be mixed with literal subscripts within an item reference; e.g., SALE (1,DEPT,2) references the second sale within the department (determined by the value of the data-name DEPT) in the first store. A single data-name may be used to refer to items in more than one table, and the tables need not have elements of the same size. Also, the same data-name may be used to reference both a one-dimensional table and another two- or three-dimensional table.

## INDEXING

Another method of specifying occurrence numbers is to affix one or more index-names to an item whose data description includes an OCCURS clause by using the optional INDEXED BY phrase.

An index-name has no separate data description entry since its definition is hardware oriented and it acts similar to an index register whose contents correspond to an occurrence number. Because of its contents, an index-name can be assigned to only one table. At object program execution, the contents of the index-name will correspond to an occurrence number for that specific dimension of the table with which the index-name was associated. An index-name must be initialized by a SET statement before being used as a table reference. Index-name values must not be less than one nor greater than the maximum occurrence number for the table element.

References are made to individual items within a table of elements by specifying the name of the item, followed by its related index-name(s) in parentheses. Each occurrence number required to complete the reference will be obtained from the respective index-name, with the index-name acting as a subscript.

When a reference requires more than one occurrence number for completeness, the user must not mix index-names and data-name subscripts in the reference. Therefore, if indexing is to be used, each OCCURS clause within the hierarchy (each dimension of the table) must contain an INDEXED BY phrase. The user may, however, mix literals and index-names within one reference, just as literals and data-name subscripts may also be mixed.

An index-name cannot be defined as part of a file; therefore, the index-name cannot be manipulated by input-output statements. Its value can be modified only by PERFORM, SEARCH, and SET statements. However, a data item in a file can be described with USAGE INDEX; transfers may then be made, without conversion, between these data items and index-names using the SET statement. Such data items are called index data items.

Direct indexing is the use of an index-name in the form of a subscript. Relative indexing may also be specified by following the index-name with an operator (+ or -) and an integer. (A space must precede and follow the operator.) The occurrence number, then, is the same as if the integer were added to or subtracted from the occurrence number to which the setting of the index-name corresponds. Relative indexing does not cause altering of the index-name value.

For example, in the reference

```
STORE (NAME + 2)
```

the index-name NAME does not change in value; however, if its value were set at the third occurrence of STORE, the fifth occurrence will actually be addressed. The advantage of relative indexing is that many different table elements may be referenced without changing the value of the index-name. When necessary, the value of an index-name may be temporarily stored without conversion into an index data item.

## Rules for Subscripting and Indexing

1. Index-names may not be combined with data-name subscripts in a single data-name reference.
2. Where subscripting is not permitted, indexing is also not permitted.
3. Tables may have one, two, or three dimensions. Thus, a reference to a table element may require up to three subscripts or index-names.
4. The use of a data-name subscript in any reference to a table element or to an item within a table element will not cause any index-name to be altered by the object program.
5. A data-name may neither be subscripted nor indexed when the data-name is itself used as a subscript or index-name in that particular reference.
6. An occurrence number specified by a subscript or implied by an index-name must not be less than one or greater than the highest permissible occurrence number for the table element.
7. When indexing is used to reference a table, the INDEXED BY phrase must be employed in each OCCURS clause used to define the table. The SEARCH and SET statements appear in the Procedure Division only in connection with indexed table references.

## Subscripting and Indexing - Sample Problem

To clarify the difference between subscripting and indexing techniques, a simple table handling problem will be solved by each method. Consider the following two tables:

```
01 LIAB-RATES.
 02 TERR-L OCCURS 9 TIMES.
 03 PREM PIC IS 9(6).
 03 CLAS OCCURS 7 TIMES.
 04 L-LIMIT PIC 9(6) OCCURS 5 TIMES.

01 DAMG-RATES.
 10 TERR-D OCCURS 9 TIMES.
 15 COMP-50D PIC IS 9(6).
 15 COLL-50D PIC IS 9(6).
 15 COMPOSIT PIC 9(6) OCCURS 196 TIMES.
```

These two tables are simplified versions of what might be used in developing automobile insurance premiums. If it is known that the territorial definition is the same for both liability and damage premiums, then both sets of territorial values can be referenced by the same data-name subscript.

Assume that an input file record of an individual policy is to be updated, and the input record contains the following data:

TERR-IN  
CLASS-IN  
LIMIT-IN

CODE-IN  
AGE-IN

where: TERR-IN is the territory number; CLASS-IN is a class code in the range 1 to 7; and LIMIT-IN is a coverage limit code in the range 1 to 5.

CODE-IN and AGE-IN are code numbers which, when multiplied, produce an occurrence number which points to 1 of 196 composite items (effectively, a table within a table).

Although the two tables have different dimensions, TERR-IN can be used to refer to the proper territorial occurrence in either table.

Example:

L-LIMIT (TERR-IN, CLASS-IN, LIMIT-IN) or  
TERR-D (TERR-IN)

are both valid uses of the contents of TERR-IN as a subscript.

The following procedure is also valid:

MULTIPLY CODE-IN BY AGE-IN GIVING SCRATCH.

MULTIPLY COMPOSIT (TERR-IN, SCRATCH)  
BY COLL-50D (TERR-IN)  
GIVING COLL-PREMIUM.

The first multiplication places the appropriate occurrence number for COMPOSIT into SCRATCH. Then the proper COMPOSIT element is multiplied by the \$50-deductible collision rate for the territory to get a final collision premium into COLL-PREMIUM.

The above techniques describe some of the organizational and technical aspects of handling tables by the subscripting method.

If indexing rather than subscripting were used with the same table formats, the tables might be defined as follows:

```
01 LIAB-RATES.
 02 TERR-L OCCURS 9 TIMES INDEXED BY XTL.
 03 PREM PIC IS 9(6).
 03 CLAS OCCURS 7 TIMES INDEXED BY XCD.
 04 L-LIMIT PIC 9(6) OCCURS 5 TIMES
 INDEXED BY XLF.

01 DAMG-RATES.
 10 TERR-D OCCURS 9 TIMES INDEXED BY XTD.
 15 COMP-50D PIC IS 9(6).
 15 COLL-50D PIC IS 9(6).
 15 COMPOSIT PIC 9(6) OCCURS 196 TIMES
 INDEXED BY XCF.
```

Indexing cannot be used to reference a table element unless the INDEXED BY phrase appears in the data description of the item and in any table group items to which the table element is subordinate. For example, COMPOSIT is INDEXED BY XCF. Thus, TERR-D (which has nine occurrences, each containing 196 occurrences of COMPOSIT) must also be indexed.

The handling of tables by indexing is similar to subscripting, except that index-names must be modified and controlled by means of SET and SEARCH statements. Data items that were used as subscripts in the subscripting example (such as TERR-IN, LIMIT-IN, SCRATCH, etc.) would have to be converted by means of a SET statement.

Example:

```
SET XTL, XTD TO TERR-IN.
```

If the data-name TERR-IN is described by a USAGE IS INDEX clause, it is moved without conversion to index-names XTL and XTD. If TERR-IN is a data-name not described by a USAGE IS INDEX clause, the compiler interprets the value in TERR-IN as a subscript value which must be converted to the appropriate index value.

#### Size Restriction Upon Character-Oriented Arrays

Because of the hardware restriction of 4095 as the largest character count in any sequence character operation, statements must not reference any part of a character array that exceeds this restriction. Greater efficiency and nearly unlimited lengths may be achieved by word orienting each array. This may be done by either using the SYNCHRONIZED clause at the occurs level or by making the size of each array a multiple of six characters. This restriction does not limit a character array to 4095 characters, but does limit references to arrays or fields whose sizes exceed 4095 characters. A warning message will be issued if any references to these fields are made.

Another limitation on arrays may require splitting very large records (01 levels) into several smaller records. The maximum number of characters that may be allocated space in one record is 262,143.

## SEARCH STATEMENT

The SEARCH statement is used to search a table for a table element that satisfies the specified condition and to adjust the associated index-name to indicate that table element.

In both Formats 1 and 2, identifier-1 must not be subscripted or indexed, but its description must contain an OCCURS clause and an INDEXED BY phrase. The description of identifier-1 in Format 2 must also contain the KEY IS option in its OCCURS clause.

Identifier-2, when specified, must be described with USAGE INDEX or as a numeric elementary item with no positions to the right of the assumed decimal point. Identifier-2 is incremented by the same amount as, and at the same time as, the occurrence number represented by the index-name associated with identifier-1 is incremented.

In Format 1, condition-1, condition-2, etc., may be any condition as defined in Section XII, Conditional Procedures.

In Format 2, condition-1 may consist of a relation condition incorporating the relation EQUALS or EQUAL TO or equal sign, or a condition-name condition, where the VALUE clause that describes the condition-name contains only a single literal. Alternatively, condition-1 may be a compound condition formed from simple conditions of the type just mentioned, with AND as the only connective. Any data-name that appears in the KEY phrase of identifier-1 may appear as the subject or object of a test or be the name of the conditional variable with which the tested condition-name is associated; however, all preceding data-names in the KEY phrase must also be included within condition-1. No other tests may appear within condition-1.

If Format 1 of the SEARCH statement is used, a serial type of search operation takes place, starting with the current index setting. If the VARYING phrase specifying index-name-1 is used and index-name-1 occurs in the INDEXED BY phrase associated with identifier-1, index-name-1 specifies the index which controls the execution of the SEARCH statement. If the VARYING phrase is not used or does not specify an index-name-1 which occurs in the INDEXED BY phrase associated with identifier-1, the index which controls the execution of the SEARCH statement is specified by the first index-name that appears in the INDEXED BY phrase associated with identifier-1.

- a. If, at the start of execution of the SEARCH statement, the index-name associated with identifier-1 contains a value that corresponds to an occurrence number that is greater than the highest permissible occurrence number for identifier-1, the SEARCH is terminated immediately. Then, if the AT END phrase is specified, imperative-statement-1 is executed; if the AT END phrase is not specified, control passes to the next sentence.

- b. If, at the start of execution of the SEARCH statement, the index-name associated with identifier-1 contains a value that corresponds to an occurrence number that is not greater than the highest permissible occurrence number for identifier-1, the SEARCH statement operates by evaluating the conditions in the order that they are written, making use of the index settings, wherever specified, to determine the occurrence of those items to be tested. If none of the conditions are satisfied, the index-name for identifier-1 is incremented to obtain reference to the next occurrence. The process is then repeated, using the new index-name settings unless the new value of the index-name settings for identifier-1 corresponds to a table element which exceeds the last element of the table by one or more occurrences, in which case the search terminates as indicated in a. above. If one of the conditions is satisfied upon its evaluation, the search terminates immediately and the imperative-statement associated with that condition is executed; the index-name remains set at the occurrence which caused the condition to be satisfied.

If Format 2 of the SEARCH statement is used, a nonserial type of search operation takes place, in which case the initial setting of the index-name for identifier-1 is ignored and its setting is varied during the search operation in a manner which allows a 'binary' search operation to be executed, with the restriction that at no time is it set to a value that exceeds the value which corresponds to the last element of the table, or that is less than the value that corresponds to the first element of the table. The index that controls the execution of the SEARCH statement is specified by the first index-name that appears in the INDEXED BY phrase associated with identifier-1. If condition-1 cannot be satisfied for any setting of the index within this permitted range, control is passed to imperative-statement-1 when the AT END phrase appears, or to the next sentence when the AT END phrase does not appear; in either case the final setting of the index is not predictable. If condition-1 can be satisfied, the index indicates an occurrence that allows condition-1 to be satisfied, and control passes to imperative-statement-2.

After the execution of an imperative-statement-1, imperative-statement-2, or imperative-statement-3 that does not terminate with a GO TO statement, control passes to the next sentence.

In the VARYING index-name-1 phrase, if index-name-1 appears in the INDEXED BY phrase of another table entry, the occurrence number represented by index-name-1 is incremented by the same amount as, and at the same time as, the occurrence number represented by the index-name associated with identifier-1 is incremented.

If identifier-1 is a data item subordinate to a data item that contains an OCCURS clause (providing for a two- or three-dimensional table), an index-name must be associated with each dimension of the table through the INDEXED BY phrase of the OCCURS clause. Only the setting of the index-name associated with identifier-1 (and the data item identifier-2 or index-name-1, if present) is modified by the execution of the SEARCH statement. To search an entire two- or three-dimensional table, it is necessary to execute a SEARCH statement several times. Prior to each execution of a SEARCH statement, SET statements must be executed whenever index-names must be adjusted to appropriate settings.

## SET STATEMENT

The SET statement is used to establish reference points for table handling operations by setting index-names to values associated with table elements.

In the following rules for the SET statement, 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.

### SET Statement Rules

1. All identifiers must name either index data items or elementary items described as integers, except that identifier-4 in Format 2 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 uniquely defined by being specified in the INDEXED BY phrase of the OCCURS clause.
2. In Format 1, the following action occurs:
  - a. Index-name-1 is set to a value causing it to refer to the table element that corresponds in occurrence number to the table element referred to by index-name-3, identifier-3, or literal-1. If identifier-3 is an index data item, or if index-name-3 is related to the same table as index-name-1, no conversion takes place. If the value contained in an index data item does not correspond to an occurrence number of an element in the table indexed by index-name-1, the result is undefined.
  - b. If identifier-1 is an index data item, it may be set equal to either the contents of index-name-3 or identifier-3, where identifier-3 is also an index data item. Literal-1 cannot be used in this case.
  - c. If identifier-1 is not an index data item, it may be set only to an occurrence number that corresponds to the value of index-name-3. Neither identifier-3 nor literal-1 can be used in this case.
  - d. The process is repeated for index-name-2, identifier-2, etc., if specified. Each time, the value of index-name-3 or identifier-3 is used as it was at the beginning of the execution of the statement. Any subscripting or indexing associated with identifier-1, etc., is evaluated immediately before the value of the respective data item is changed.
3. 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; thereafter, the process is repeated for index-name-5, etc. Each time, the value of identifier-4 is used as it was at the beginning of the execution of the statement.
4. Data in the following chart represents the validity of various operand combinations in the SET statement. The numeric reference indicates the applicable rule (above).



| Sending Item      | Receiving Item    |                       |                       |
|-------------------|-------------------|-----------------------|-----------------------|
|                   | Integer Data Item | Index-Name            | Index Data Item       |
| Integer Literal   | Invalid/2c        | Valid/2a              | Invalid/2b            |
| Integer Data Item | Invalid/2c        | Valid/2a              | Invalid/2b            |
| Index-Name        | Valid/2c          | Valid/2a              | Valid/2b <sup>1</sup> |
| Index Data Item   | Invalid/2c        | Valid/2a <sup>1</sup> | Valid/2b <sup>1</sup> |

Comparisons Involving Index-Names and/or Index Data Items

The comparison of two index-names is the same as if the corresponding occurrence numbers were compared. Similarly, when an index-name is compared with a data item (other than an index data item) or a literal, it is the same as if their corresponding occurrence numbers were compared.

When a comparison between an index-name and an index data item is made, the actual values are compared without conversion. Any other comparison involving an index data item is invalid and will cause unpredictable results.

SEARCH and SET Statement Examples

Some examples of SEARCH and SET statement usage are presented below.

**Example:**

SET IX-TABLE TO 3.

The index-name IX-TABLE is set to an index value that corresponds to occurrence number three for that table. If index-name IX-TABLE is not defined using an INDEXED BY phrase, the statement is illegal.

**Example:**

SET LOOKUP TO KEY-POINT.

---

<sup>1</sup> No conversion takes place.

If LOOKUP is an index-name, it is set to the occurrence number that corresponds to KEY-POINT. If KEY-POINT is an index-name that is not related to the same table as LOOKUP, an appropriate conversion is performed; otherwise, no conversion takes place. If LOOKUP is an index data item (that is, defined with a USAGE INDEX clause), it is set to the actual contents of KEY-POINT. KEY-POINT must be either an index data item or an index-name or the statement is illegal.

If LOOKUP is neither an index-name nor an index data item, KEY-POINT must be an index-name. LOOKUP is then set to the occurrence value to which index-name KEY-POINT corresponds.

Example:

```
SET DEPT-INDEX UP BY KEY-JUMP.
```

The index-name DLPT-INDEX is incremented by a value that corresponds to the number of occurrences indicated by KEY-JUMP. That is, if the value of KEY-JUMP is three, DEPT-INDEX is incremented by a value that is equivalent to three occurrences.

Consider a hypothetical table of wholesale discount factors involved in determining the total price of merchandise orders, such as:

```
01 DISCOUNT-TABLE.
02 RANGE-ENTRY, OCCURS 16 TIMES INDEXED BY XRANGE.
03 MAXRANGE PIC IS 9(6).
03 CLASS-ITEM OCCURS 12 TIMES INDEXED BY XCLASS PIC 9(6).
```

Each wholesale order input record is computed against a basic price schedule catalog. An item in the wholesale order record, TOTAL-PRICE, is then compared against the discount schedule to determine which set of discount rates is to be applied to the particular order. Each occurrence of RANGE-ENTRY contains a field called MAXRANGE and 12 occurrences of CLASS-ITEM. MAXRANGE specifies the maximum order amount for which the accompanying set of CLASS-ITEM discount factors can be applied. Each CLASS-ITEM contains a discount factor for a particular class of merchandise. The coding could be:

```
SET XRANGE TO 1.

SEARCH RANGE-ENTRY AT END GO TO ERROR WHEN MAXRANGE
(XRANGE) > TOTAL-PRICE GO TO RANGE-LOCATED.
```

When the search is completed, XRANGE will be set at the occurrence of RANGE-ENTRY that contains the appropriate set of discount factors.

As another example, suppose there are two tables which contain calendar information for the 12 months of the year. One table, JULIAN-VALUES, contains the Julian day number for the first day of each month; another table, INDEX-TABLE, contains month names ordered by ascending key of month numbers, as follows:

```
01 JULIAN-VALUES.
02 JULIANS PIC X(36) VALUE
"001032060091121152182213244274305335".
02 JULIAN-TABLE REDEFINES JULIANS
OCCURS 12 TIMES INDEXED BY X2.
03 FIRST-JULIAN PIC 999.
```

```

01 INDEX-TABLE.
02 CALENDAR.
03 QUARTER-1 PIC X(21) VALUE
 "JAN0131FEB0228MAR0331".
03 QUARTER-2 PIC X(21) VALUE
 "APR0430MAY0531JUN0630".
03 QUARTER-3 PIC X(21) VALUE
 "JUL0731AUG0831SEP0930".
03 QUARTER-4 PIC X(21) VALUE
 "OCT1031NOV1130DEC1231".
02 CAL-TABLE REDEFINES CALENDAR
OCCURS 12 TIMES INDEXED BY X1
ASCENDING KEY IS MONTH-NUM.
03 CAL-ITEM.
04 MONTH PIC XXX.
04 MONTH-NUM PIC 99.
04 MAX-DAYS PIC 99.

```

If each input record contains some month number, INPUT-MONTH-NO, and the matching month and beginning Julian day number are to be obtained from the tables for reporting purposes, either a serial or binary search can be used.

For a binary search (Format 2), the example procedures are:

```

SEARCH ALL CAL-TABLE AT END DISPLAY
 "BAD INPUT MONTH" GO TO ERROR-RTN
WHEN MONTH-NUM (X1) = INPUT-MONTH-NO
MOVE MONTH (X1) TO REPORT-MONTH
SET X2 TO X1
MOVE FIRST-JULIAN (X2) TO BEGIN-JULIAN-DATE.

```

The index-name X1 requires no SET statement prior to a binary search since it is implicitly set upon entering the search operation. However, it is necessary that the table, CAL-TABLE, be ordered on a key, MONTH-NUM in the example, in order to use Format 2 of the SEARCH statement. When the condition is satisfied, index-name X1 is left pointing to the table element which met the condition. Therefore, its value may be used in a SET statement to adjust index-name X2 so that a table element corresponding to the match in CAL-TABLE may be obtained from JULIAN-TABLE.

The advantage of the Format 2 SEARCH statement is the relative speed of operation which increases as the size of the tables to be searched increases. However, with Format 2, data must be arranged in order of KEY values. Also, the results of the binary search are unpredictable if a table contains any duplicate items or items that are out of sequence.

The serial search technique (Format 1) may be used if a binary search is not practical. It would be slower but, in the case of duplicate items, it would be possible to locate all table elements which satisfy a condition by continuing the search operation after first meeting the condition and then setting the index-name up by 1.

The previous example with a serial search (Format 1) is:

```

SET X1, X2 TO 1.
SEARCH CAL-TABLE VARYING X2 AT END DISPLAY
 "BAD INPUT MONTH" GO TO ERROR-RTN
WHEN MONTH-NUM (X1) = INPUT-MONTH-NO
MOVE MONTH (X1) TO REPORT-MONTH
MOVE FIRST-JULIAN (X2) TO BEGIN-JULIAN-DATE.

```

## SECTION XIV

### LIBRARY FACILITY

#### DESCRIPTION OF THE LIBRARY FACILITY

The library feature provides the capability for specifying text that is to be copied from a library.

COBOL libraries contain source text material that is available to the compiler for copying when the program is being compiled. A short phrase utilizing the COPY statement can cause large portions of source library text to be inserted into the source program where it is treated by the compiler as part of the source program, thus eliminating repetitious coding. Once established, a source library may be referenced many times by many programs.

COBOL library text is placed on the COBOL library as an operation independent of the COBOL program. More than one COBOL library may be available at program compilation.

#### COPY FUNCTIONS

COBOL provides two distinct and mutually exclusive COPY functions. The first, referred to as HIS COPY, is the process that has been available in the earlier (prestandard) version of COBOL. The second function, referred to as American National Standard COPY, represents Level 2 of the American National Standard COBOL X3.23-1968 library facility.

#### HIS COPY

To use the HIS COPY function, the COPY option must be included on the \$ COBOL control card at program compilation. This option allows the user to copy source lines from an external library, utilize internal program copies, and use the RENAMING phrase in the FILE-CONTROL paragraph of the Environment Division.

Data Division source lines may be repeated within a program by copying a record-name or group-name. This operation is called an internal copy procedure and may be accomplished using the following syntax:

```
level-number data-name-1 COPY data-name-2.
```

Data-name-2 specifies a record-name or a group-name. Refer to Figure 14-2 for an example of the internal copy function.

COBOL source statements may be inserted into a program from the user library file (.L) at program compilation by specifying either of the following in a source program:

The Data Division clause

$\left. \begin{array}{l} \text{level indicator} \\ \text{level-number} \end{array} \right\} \text{data-name-1 } \underline{\text{COPY}} \text{ data-name-2 } [ \text{FROM } \underline{\text{LIBRARY}} ] .$

The Procedure Division statement

paragraph-name. COPY library-name FROM LIBRARY .

When a user library file is created, the preparation of lengthy repetitions of file, report, and record descriptions for the Data Division and paragraphs for the Procedure Division may be avoided in programs that use common data or procedures.

#### HIS COPY Source Library Format

The HIS COPY library consists of syntactically correct Data and Procedure Division statements in card/line image format. The library entries to be inserted into the source program must be present on a user library file (.L) at program compilation. Library entries to be inserted into the Data Division begin with a line containing a level indicator (FD, SD, RD) or a level-number (01, 66, 77, 88) starting in column 8. Library entries to be inserted into the Procedure Division begin with a line containing a library-name in column 8. Refer to Figure 14-1 for an example of a HIS COPY library.

Within the Data Division, the HIS COPY...FROM LIBRARY clause depends upon a match between data-name-2 as specified in the COPY clause and a corresponding data-name defined on the library. When a match occurs, an additional check is made to determine the level-number relationship between the source line containing the COPY clause and the matching library data-name. If the level-numbers are the same, the library text is copied unchanged. If the level-numbers are not equal, the COPY processor attempts to internally adjust the level-numbers of the copied text by an amount equal to the difference between the level-number of data-name-1 in the COPY clause and the level-number of the matching library data-name. This procedure is transparent to the user since the 'adjusted' level-numbers are not reflected in the compiled source listing. Reliance upon this adjustment feature is not recommended since it may result in an unusable level structure.

Within the Data Division, library text is copied beginning with the library entry containing the data-name on which the match occurred and continuing until a level-number is encountered that is equal to or less than the level-number of the matching library data-name. If the library text to be copied is an FD, SD, or RD entry, the copy procedure is terminated by the next level indicator or level-number encountered. The copy procedure may also be terminated when the Procedure Division header entry is encountered on the library.

Within the Procedure Division, library text is copied beginning with the library entry containing the library-name on which a match occurs and continuing until the next library-name is encountered. On the COBOL library, library-name is equivalent to paragraph-name in the HIS COPY format. See Figures 14-1, 14-2, and 14-3.

The HIS COPY feature may be used to copy library lines that are subordinate to 01 level entries. However, a COPY statement specifying a subordinate level data-name must include qualification of that data-name by its corresponding 01 level library data-name even though such qualification does not appear to be necessary to ensure a unique reference. For example, Figure 14-2 contains the statement

```
02 WORK-ACCNT COPY ACCNT-NO OF LIB-REC-2 FROM LIBRARY.
```

in which ACCNT-NO must be qualified by LIB-REC-2 in order to satisfy HIS COPY requirements.

When Procedure Division statements refer to copied text that requires qualification, the highest level qualifier is that data-name-1 defined in the source line containing the COPY statement. The library data-name on which the match occurred, although copied into the source program, must not be used for qualification. For example, Figure 14-2 contains the statement

```
MOVE ... TO ACCOUNT-NO OF WORK-REC-1 ...
```

in which ACCOUNT-NO is qualified by WORK-REC-1, not by the library data-name LIB-REC-1.

The following rules are applicable to the HIS COPY library format:

1. COPY library text contains two types of entries:
  - a. Data Division entries.
  - b. Procedure Division entries.

All Data Division text entries must appear on the library before the first Procedure Division entry and must be separated from the first Procedure Division entry by a line containing a Procedure Division header starting in column 8.

2. Each Data Division library text entry starts with a line containing a level indicator (FD, SD, RD) or a level-number (01, 66, 77, 88) beginning in column 8 and continues until the next such line is encountered. Lines beginning with level-numbers that are not in column 8 neither start nor terminate a library entry.
3. A Data Division library text entry may be partially copied by specifying a data-name-2 that matches a library data-name subordinate to a level 01 entry. In this instance, the library text is copied until a level-number is encountered which is equal to or less than the level-number of the matched library data-name.
4. If the library contains only Data Division library text entries, the library must be terminated by a Procedure Division header.

5. A Procedure Division library text entry starts with a line containing a library-name beginning in column 8 and followed by a period. A Procedure Division library text entry continues until the next library-name beginning in column 8 is encountered.
6. Library-names defined in the HIS COPY library must be unique.
7. If the library contains only Procedure Division library text entries, these entries must be preceded by a line containing a Procedure Division header beginning in column 8.
8. If the HIS COPY library contains any Procedure Division entries, they must be terminated with a dummy ending library-name beginning in column 8.
9. No COPY statements may appear in the HIS library text.
10. If format or syntax errors are encountered in source lines copied from a library, compilation results are unpredictable.
11. If a record description containing multiple REDEFINES clauses is copied from a library into FD or SD entries that are not defined in the same order as the corresponding SELECT statements, error messages may result.

#### Reference Listing Format

The result of merging the library or internal copy information may be observed on the reference listing by locating those lines under the heading REF LINE # which are of the form Cnnnnn, where each satisfied COPY starts with C00001, C00002, ..., Cnnnnn. Copied lines are cross-referenced in the COMPILER COMMENTS column in the form 'REF TO aaaaa (Cnnnnn)', where aaaaa is the alter number and nnnnn is the number of the copied line that follows the alter number.

#### Missing Library-Name

If a library-name specified in a COPY ... FROM LIBRARY statement does not appear on the associated library, the following error message is printed:

```
***** COPY OR RENAMING OBJECT UNDEFINED -- DELETED REFERENCE
```

#### Compressed Deck Options

The compiler supports two compressed deck options, COMDK and CCOMDK. The CCOMDK option allows the user to indicate that copied library text is to be included in the compressed deck. Refer to the Use of Compressed Decks paragraph in Section XVI.

## HIS COPY WITH COMDK

If the COPY and COMDK options are specified on the \$ COBOL card, the copied text (from the library, an internal copy, or a RENAMING phrase) is not included in the resultant compressed deck. The reference listing will show no COBOL alter numbers (columns 1-5) for the copied text, which is identified by a 'C' prefix in the reference line number field. The compressed deck may be altered for subsequent compilations by using the COBOL alter number field. See Figures 14-1 and 14-2.

## HIS COPY WITH CCOMDK

If the COPY and CCOMDK options are specified on the \$ COBOL card, the copied text as well as the COPY statement itself is included in the resultant compressed deck. The reference listing will show corresponding COBOL alter numbers for each of the copied statements, which are identified by a 'C' prefix in the reference line number field. To use the resultant compressed deck in subsequent compilations, it is necessary to remove the statement containing the COPY clause, regardless of whether it is an internal data description COPY, a COPY FROM LIBRARY statement, or a RENAMING phrase in a FILE-CONTROL paragraph. This can be accomplished by utilizing the COBOL alter number field. Specification of the COPY and CCOMDK options together is discouraged due to the complexity of subsequent compressed deck preparation and usage. See Figures 14-1 and 14-3.

```
1 8 12

$ DATA .L
01 LIB-REC-1.
 03 ACCOUNT-NO PIC X(20).
 03 BALANCE PIC 9(16)V99 VALUE 0.
01 LIB-REC-2.
 02 ACCOUNT-NO.
 03 SUB-1 PIC X(10).
 03 SUB-2 PIC X(10).
 02 BALANCE PIC $,,$,$,$,$,$,$,$,$,$9.99 VALUE 0.
 02 ACCNT-NO.
 03 SUB-3 PIC X(10).
 03 SUB-4 PIC X(10).
PROCEDURE DIVISION.
LIB-PARA.
 OPEN INPUT MASTER-FILE.
 MOVE SPACES TO WORK-ACCOUNT-NO.
DUMMY-PARAGRAPH-NAME.
```

Figure 14-1. Library for Figures 14-2 and 14-3



| COBOL<br>ALT # | SOURCE LISTING                                                                 | REF<br>LINE # | COMPILER COMMENTS                                               |
|----------------|--------------------------------------------------------------------------------|---------------|-----------------------------------------------------------------|
| 00001          | IDENTIFICATION DIVISION.                                                       | 00001         |                                                                 |
| 00002          | PROGRAM-ID. HISCP1.                                                            | 00002         |                                                                 |
| 00003          | ENVIRONMENT DIVISION.                                                          | 00003         |                                                                 |
| 00004          | CONFIGURATION SECTION.                                                         | 00004         |                                                                 |
| 00005          | SPECIAL-NAMES.                                                                 | 00005         |                                                                 |
| 00006          | INPUT-OUTPUT SECTION.                                                          | 00006         |                                                                 |
| 00007          | FILE-CONTROL.                                                                  | 00007         |                                                                 |
| 00008          | SELECT MASTER-FILE ASSIGN TO A1.                                               | 00008         |                                                                 |
| 00009          | DATA DIVISION.                                                                 | 00009         |                                                                 |
|                |                                                                                | ***           | WA OBJECT-COMPUTER PARAGRAPH MISSING--<br>ASSUMED 6000 WITH EIS |
| 00010          | FILE SECTION.                                                                  | 00010         |                                                                 |
| 00011          | *EJECT                                                                         | 00011         |                                                                 |
| 00012          | FD MASTER-FILE LABEL RECORDS ARE STANDARD.                                     | 00012         |                                                                 |
| 00013          | 01 REC-1.                                                                      | 00013         | CONTAINS 60 CHARACTERS 10 WORDS                                 |
| 00014          | 02 NAME PIC X(38).                                                             | 00014         | STARTS IN CHARACTER POSITION 1                                  |
| 00015          | 02 FILLER PIC XX.                                                              | 00015         | STARTS IN CHARACTER POSITION 39                                 |
| 00016          | 02 ACCOUNT-NO PIC X(20).                                                       | 00016         | STARTS IN CHARACTER POSITION 41                                 |
| 00017          | 01 REC-2 COPY REC-1.                                                           | 00017         | REF BY 00030 CONTAINS 80 CHARACTERS 14 WORDS                    |
|                | 01 REC-1.                                                                      | C00001        |                                                                 |
|                | 02 NAME PIC X(38).                                                             | C00002        | STARTS IN CHARACTER POSITION 1                                  |
|                | 02 FILLER PIC XX.                                                              | C00003        | STARTS IN CHARACTER POSITION 39                                 |
|                | 02 ACCOUNT-NO PIC X(20).                                                       | C00004        | STARTS IN CHARACTER POSITION 41                                 |
|                | 02 DESCRIPTION PIC X(20).                                                      | 00018         | STARTS IN CHARACTER POSITION 61                                 |
| 00018          | WORKING-STORAGE SECTION.                                                       | 00019         |                                                                 |
| 00019          | 77 WORK-ACCOUNT-NO PIC X(20).                                                  | 00020         | REF BY 00028                                                    |
| 00020          | 01 WORK-REC-1 COPY LIB-REC-1 FROM LIBRARY.                                     | 00021         | CONTAINS 76 CHARACTERS 13 WORDS                                 |
| 00021          | 01 LIB-REC-1.                                                                  | C00001        |                                                                 |
|                | 03 ACCOUNT-NO PIC X(20).                                                       | C00002        | STARTS IN CHARACTER POSITION 1                                  |
|                | 03 BALANCE PIC 9(16)V99 VALUE 0.                                               | C00003        | REF BY 00030 STARTS IN CHARACTER POSITION 21                    |
| 00022          | 03 NAME PIC X(38) VALUE SPACES.                                                | 00022         | STARTS IN CHARACTER POSITION 39                                 |
| 00023          | 01 WORK-REC-2 COPY LIB-REC-2 FROM LIBRARY.                                     | 00023         | CONTAINS 64 CHARACTERS 11 WORDS                                 |
|                | 01 LIB-REC-2.                                                                  | C00001        |                                                                 |
|                | 02 ACCOUNT-NO.                                                                 | C00002        | REF BY 00030 STARTS IN CHARACTER POSITION 1                     |
|                | 03 SUB-1 PIC X(10).                                                            | C00003        | STARTS IN CHARACTER POSITION 11                                 |
|                | 03 SUB-2 PIC X(10).                                                            | C00004        | STARTS IN CHARACTER POSITION 21                                 |
|                | 02 BALANCE PIC \$, \$\$\$, \$\$\$, \$\$\$, \$\$\$, \$\$\$, \$\$\$9.99 VALUE 0. | C00005        | STARTS IN CHARACTER POSITION 45                                 |
|                | 02 ACCNT-NO.                                                                   | C00006        | STARTS IN CHARACTER POSITION 55                                 |
|                | 03 SUB-3 PIC X(10).                                                            | C00007        | CONTAINS 20 CHARACTERS 4 WORDS                                  |
|                | 03 SUB-4 PIC X(10).                                                            | C00008        |                                                                 |
| 00024          | 01 WORK-REC-3.                                                                 | 00024         |                                                                 |
| 00025          | 02 WORK-ACCNT COPY ACCNT-NO OF LIB-REC-2 FROM LIBRARY.                         | 00025         |                                                                 |
|                | 02 ACCNT-NO.                                                                   | C00001        |                                                                 |
|                | 03 SUB-3 PIC X(10).                                                            | C00002        | STARTS IN CHARACTER POSITION 1                                  |
|                | 03 SUB-4 PIC X(10).                                                            | C00003        | STARTS IN CHARACTER POSITION 11                                 |
| 00026          | PROCEDURE DIVISION.                                                            | 00026         |                                                                 |
| 00027          | START SECTION.                                                                 | S00027        |                                                                 |
| 00028          | PARA-1. COPY LIB-PARA FROM LIBRARY.                                            | P00028        |                                                                 |
|                | LIB-PARA.                                                                      | C00001        | REF TO 00020                                                    |
|                | OPEN INPUT MASTER-FILE.                                                        | C00002        |                                                                 |
|                | MOVE SPACES TO WORK-ACCOUNT-NO.                                                | C00003        |                                                                 |

14-6

DD26

Figure 14-2. HIS COPY WITH COMDK Option

| COBOL<br>ALT # | S O U R C E   L I S T I N G                          | REF<br>LINE # | C O M P I L E R | C O M M E N T S           |
|----------------|------------------------------------------------------|---------------|-----------------|---------------------------|
| 00029          | PARA-2. READ MASTER-FILE AT END GO TO DONE.          | P00029        |                 |                           |
| 00030          | MOVE ACCOUNT-NO OF REC-1 TO ACCOUNT-NO OF WORK-REC-1 | 00030         | REF TO 00032    |                           |
| 00031          | ACCOUNT-NO OF WORK-REC-2.                            | 00031         | REF TO 00016    | 00021 C00C02 00023 C00002 |
| 00032          | DONE. STOP RUN.                                      | P00032        |                 |                           |
| 00033          | END PROGRAM.                                         | 00033         | REF BY 00029    |                           |

\*\*\*\*\* THE ABOVE LISTING CONTAINS 000 ERROR MESSAGES \*\*\*\*\*

\*\*\* THE ABOVE LISTING CONTAINS 001 WARNING MESSAGES \*\*\*

\* THE ABOVE LISTING CONTAINS 000 EFFICIENCY MESSAGES \*

COMPILATION TIME (MIN): ELAP CLOCK= 000.22      PROC= 000.03

00000 OVERFLOW READS      00000 OVERFLOW WRITES      22475 WORDS MEMORY USED      00003 LINKS USED ON \*3 FILE

14-7

DD26

Figure 14-2. HIS COPY WITH COMDK Option (cont.)

COBOL  
ALT #

SOURCE LISTING

REF COMPILER COMMENTS  
LINE #

```
00001 IDENTIFICATION DIVISION.
00002 PROGRAM-ID. HISCP2.
00003 ENVIRONMENT DIVISION.
00004 CONFIGURATION SECTION.
00005 SPECIAL-NAMES.
00006 INPUT-OUTPUT SECTION.
00007 FILE-CONTROL.
00008 SELECT MASTER-FILE ASSIGN TO A1.
00009 DATA DIVISION.

00010 FILE SECTION.
00011 *EJECT

00012 FD MASTER-FILE LABEL RECORDS ARE STANDARD.
00013 01 REC-1.
00014 02 NAME PIC X(38).
00015 02 FILLER PIC XX.
00016 02 ACCOUNT-NO PIC X(20).

00017 01 REC-2 COPY REC-1.
00018 01 REC-1.
00019 02 NAME PIC X(38).
00020 02 FILLER PIC XX.
00021 02 ACCOUNT-NO PIC X(20).
00022 02 DESCRIPTION PIC X(20).
00023 WORKING-STORAGE SECTION.
00024 77 WORK-ACCOUNT-NO PIC X(20).
00025 01 WORK-REC-1 COPY LIB-REC-1 FROM LIBRARY.
00026 01 LIB-REC-1.
00027 03 ACCOUNT-NO PIC X(20).

00028 03 BALANCE PIC 9(16)V99 VALUE 0.
00029 03 NAME PIC X(38) VALUE SPACES.
00030 01 WORK-REC-2 COPY LIB-REC-2 FROM LIBRARY.
00031 01 LIB-REC-2.
00032 02 ACCOUNT-NO.
00033 03 SUB-1 PIC X(10).
00034 03 SUB-2 PIC X(10).
00035 02 BALANCE PIC $, $$$, $$$, $$$, $$$, $$$9.99 VALUE 0.
00036 02 ACCNT-NO.
00037 03 SUB-3 PIC X(10).
00038 03 SUB-4 PIC X(10).
00039 01 WORK-REC-3.
00040 02 WORK-ACNT COPY ACNT-NO OF LIB-REC-2 FROM LIBRARY.
00041 02 ACNT-NO.
00042 03 SUB-3 PIC X(10).
00043 03 SUB-4 PIC X(10).
00044 PROCEDURE DIVISION.
00045 START SECTION.
00046 PARA-1. COPY LIB-PARA FROM LIBRARY.

00047 LIB-PARA.
00048 OPEN INPUT MASTER-FILE.
00049 MOVE SPACES TO WORK-ACCOUNT-NO.
```

```
00001
00002
00003
00004
00005
00006
00007
00008
00009
*** WA OBJECT:COMPUTER PARAGRAPH MISSING--
ASSUMED 6000 WITH EIS
00010
00011

00012
00013 CONTAINS 60 CHARACTERS 10 WORDS
00014 STARTS IN CHARACTER POSITION 1
00015 STARTS IN CHARACTER POSITION 39
00016 STARTS IN CHARACTER POSITION 41
REF BY 00030
00017 CONTAINS 80 CHARACTERS 14 WORDS
C00001
C00002 STARTS IN CHARACTER POSITION 1
C00003 STARTS IN CHARACTER POSITION 39
C00004 STARTS IN CHARACTER POSITION 41
00018 STARTS IN CHARACTER POSITION 61
00019
00020 REF BY 00028
00021 CONTAINS 76 CHARACTERS 13 WORDS
C00001
C00002 STARTS IN CHARACTER POSITION 1
REF BY 00030
C00003 STARTS IN CHARACTER POSITION 21
00022 STARTS IN CHARACTER POSITION 39
00023 CONTAINS 64 CHARACTERS 11 WORDS
C00001
C00002 REF BY 00030
C00003 STARTS IN CHARACTER POSITION 1
C00004 STARTS IN CHARACTER POSITION 11
C00005 STARTS IN CHARACTER POSITION 21
C00006
C00007 STARTS IN CHARACTER POSITION 45
C00008 STARTS IN CHARACTER POSITION 55
00024 CONTAINS 20 CHARACTERS 4 WORDS
00025
C00001
C00002 STARTS IN CHARACTER POSITION 1
C00003 STARTS IN CHARACTER POSITION 11
00026
S00027
P00028 REF TO 00020
C00001
C00002
C00003
```

14-8

DD26

Figure 14-3. HIS COPY WITH CCOMDK Option

| COBOL<br>ALT # | SOURCE LISTING                                       | REF<br>LINE # | COMPILER | COMMENTS                        |
|----------------|------------------------------------------------------|---------------|----------|---------------------------------|
| 00050          | PARA-2. READ MASTER-FILE AT END GO TO DONE.          | P00029        |          |                                 |
| 00051          | MOVE ACCOUNT-NO OF REC-1 TO ACCOUNT-NO OF WORK-REC-1 | 00030         | REF TO   | 00032                           |
| 00052          | ACCOUNT-NO OF WORK-REC-2.                            | 00031         | REF TO   | 00016 00021 C00002 00023 C00002 |
| 00053          | DONE. STOP RUN.                                      | P00032        |          |                                 |
| 00054          | END PROGRAM.                                         | 00033         | REF BY   | 00029                           |

\*\*\*\*\* THE ABOVE LISTING CONTAINS 000 ERROR MESSAGES \*\*\*\*\*

\*\*\* THE ABOVE LISTING CONTAINS 001 WARNING MESSAGES \*\*\*

\* THE ABOVE LISTING CONTAINS 000 EFFICIENCY MESSAGES \*

COMPILATION TIME (MIN): ELAP CLOCK= 000.21 PROC= 000.03

00000 OVERFLOW READS 00000 OVERFLOW WRITES 22475 WORDS MEMORY USED 00003 LINKS USED ON \*3 FILE

14-9

DD26

Figure 14-3. HIS COPY WITH CCOMDK Option (cont.)

AMERICAN NATIONAL STANDARD COPY

The LIBCPY option must be specified on the \$ COBOL card in order to engage the American National Standard COPY function.

The American National Standard COPY function and the HIS COPY function are mutually exclusive. The two functions have different library formats. If a user attempts to employ both functions, the LIBCPY option overrides the \$ COBOL COPY (i.e., HIS COPY) option.

The RENAMING phrase cannot be used in conjunction with the American National Standard COPY function.

Library Format for American National Standard COPY

The library entries to be inserted into the source program must be present on a user library file (.L) at program compilation. Two library formats are available for use with American National Standard COPY. Their usage is mutually exclusive.

In the preferred format, asterisks in columns 1-6 on the library are used to indicate that the name starting in column 8 of the same line is a library-name. Information will be copied until the next record on the library having asterisks in columns 1-6 is encountered or until the end of the library file is reached. The preferred library format allows level indicators, level-numbers, section-names, and/or paragraph-names to begin in column 8 on the library. An example of the preferred format is:

```
1 6 8

***** LIB-NAME-1.
 .
 .
***** LIB-NAME-2.
 PERFORM PARA-2.
 PARA-1. MOVE A TO B.
 PARA-2. ADD C TO D.
***** LIB-NAME-3.
```

```
$ DATA .L
***** LIB-REC-1.
 03 ACCOUNT-NO PIC X(20).
 03 BALANCE PIC 9(16)V99 VALUE 0.
***** LIB-REC-2.
 02 ACCOUNT-NO.
 03 SUB-1 PIC X(10).
 03 SUB-2 PIC X(10).
 02 BALANCE PIC $,,$,$,$,$,$,$,$,$,$9.99 VALUE 0.
***** LIB-PARA.
OPEN INPUT MASTER-FILE.
MOVE SPACES TO WORK-ACCOUNT-NO.
```

Figure 14-4. Library for Figures 14-5 and 14-6

| COBOL<br>ALT # | SOURCE LISTING                                                                 | REF<br>LINE # | COMPILER                     | COMMENTS                                                     |
|----------------|--------------------------------------------------------------------------------|---------------|------------------------------|--------------------------------------------------------------|
| 00001          | IDENTIFICATION DIVISION.                                                       | 00001         |                              |                                                              |
| 00002          | PROGRAM-ID. ANSCPI.                                                            | 00002         |                              |                                                              |
| 00003          | ENVIRONMENT DIVISION.                                                          | 00003         |                              |                                                              |
| 00004          | CONFIGURATION SECTION.                                                         | 00004         |                              |                                                              |
| 00005          | SPECIAL-NAMES.                                                                 | 00005         |                              |                                                              |
| 00006          | INPUT-OUTPUT SECTION.                                                          | 00006         |                              |                                                              |
| 00007          | FILE-CONTROL.                                                                  | 00007         |                              |                                                              |
| 00008          | SELECT MASTER-FILE ASSIGN TO A1.                                               | 00008         |                              |                                                              |
| 00009          | DATA DIVISION.                                                                 | 00009         |                              |                                                              |
|                |                                                                                | ***           | WA                           | OBJECT-COMPUTER PARAGRAPH MISSING--<br>ASSUMED 6000 WITH EIS |
| 00010          | FILE SECTION.                                                                  | 00010         |                              |                                                              |
| 00011          | *EJECT                                                                         | 00011         |                              |                                                              |
| 00012          | FD MASTER-FILE LABEL RECORDS ARE STANDARD.                                     | 00012         |                              |                                                              |
| 00013          | 01 REC-1.                                                                      | 00013         | CONTAINS                     | 60 CHARACTERS 10 WORDS                                       |
| 00014          | 02 NAME PIC X(38).                                                             | 00014         | STARTS IN CHARACTER POSITION | 1                                                            |
| 00015          | 02 FILLER PIC XX.                                                              | 00015         | STARTS IN CHARACTER POSITION | 39                                                           |
| 00016          | 02 ACCOUNT-NO PIC X(20).                                                       | 00016         | STARTS IN CHARACTER POSITION | 41                                                           |
|                |                                                                                |               | REF BY                       | 00043                                                        |
| 00017          | WORKING-STORAGE SECTION.                                                       | 00017         |                              |                                                              |
| 00018          | 77 WORK-ACCOUNT-NO PIC X(20).                                                  | 00018         | REF BY                       | 00041                                                        |
| 00019          | 01 WORK-REC-1 COPY LIB-REC-1.                                                  | 00019         | CONTAINS                     | 38 CHARACTERS 7 WORDS                                        |
|                | 03 ACCOUNT-NO PIC X(20).                                                       | 00020         | STARTS IN CHARACTER POSITION | 1                                                            |
|                |                                                                                |               | REF BY                       | 00043                                                        |
|                | 03 BALANCE PIC 9(16)V99 VALUE 0.                                               | 00021         | STARTS IN CHARACTER POSITION | 21                                                           |
| 00020          | 01 WORK-REC-1A COPY LIB-REC-1.                                                 | 00022         | CONTAINS                     | 76 CHARACTERS 13 WORDS                                       |
|                | 03 ACCOUNT-NO PIC X(20).                                                       | 00023         | STARTS IN CHARACTER POSITION | 1                                                            |
|                |                                                                                |               | REF BY                       | 00043                                                        |
|                | 03 BALANCE PIC 9(16)V99 VALUE 0.                                               | 00024         | STARTS IN CHARACTER POSITION | 21                                                           |
|                | 03 NAME PIC X(38) VALUE SPACES.                                                | 00025         | STARTS IN CHARACTER POSITION | 39                                                           |
| 00021          | 01 WORK-REC-2 COPY LIB-REC-2.                                                  | 00026         | CONTAINS                     | 44 CHARACTERS 8 WORDS                                        |
| 00022          | 02 ACCOUNT-NO.                                                                 | 00027         | REF BY                       | 00043                                                        |
|                | 03 SUB-1 PIC X(10).                                                            | 00028         | STARTS IN CHARACTER POSITION | 1                                                            |
|                | 03 SUB-2 PIC X(10).                                                            | 00029         | STARTS IN CHARACTER POSITION | 11                                                           |
|                | 02 BALANCE PIC \$, \$\$\$, \$\$\$, \$\$\$, \$\$\$, \$\$\$, \$\$\$9.99 VALUE 0. | 00030         | STARTS IN CHARACTER POSITION | 21                                                           |
| 00023          | 01 WORK-REC-2A COPY LIB-REC-2.                                                 | 00031         | CONTAINS                     | 82 CHARACTERS 14 WORDS                                       |
|                | 02 ACCOUNT-NO.                                                                 | 00032         |                              |                                                              |
|                | 03 SUB-1 PIC X(10).                                                            | 00033         | STARTS IN CHARACTER POSITION | 1                                                            |
|                | 03 SUB-2 PIC X(10).                                                            | 00034         | STARTS IN CHARACTER POSITION | 11                                                           |
|                | 02 BALANCE PIC \$, \$\$\$, \$\$\$, \$\$\$, \$\$\$, \$\$\$, \$\$\$9.99 VALUE 0. | 00035         | STARTS IN CHARACTER POSITION | 21                                                           |
|                | 02 NAME PIC X(38) VALUE SPACES.                                                | 00036         | STARTS IN CHARACTER POSITION | 45                                                           |
| 00024          | PROCEDURE DIVISION.                                                            | 00037         |                              |                                                              |
| 00025          | START SECTION.                                                                 | S00038        |                              |                                                              |
| 00026          | PARA-1. COPY LIB-PARA.                                                         | P00039        |                              |                                                              |
| 00027          | OPEN INPUT MASTER-FILE.                                                        | C00040        |                              |                                                              |
|                | MOVE SPACES TO WORK-ACCOUNT-NO.                                                | C00041        | REF TO                       | 00018                                                        |
| 00028          | PARA-2. READ MASTER-FILE AT END GO TO DONE.                                    | P00042        |                              |                                                              |
|                |                                                                                |               | REF TO                       | 00047                                                        |
| 00029          | MOVE ACCOUNT-NO OF REC-1 TO                                                    | 00043         | REF TO                       | 00016 00020 00023 00027                                      |
| 00030          | ACCOUNT-NO OF WORK-REC-1                                                       | 00044         |                              |                                                              |
| 00031          | ACCOUNT-NO OF WORK-REC-1A                                                      | 00045         |                              |                                                              |
| 00032          | ACCOUNT-NO OF WORK-REC-2.                                                      | 00046         |                              |                                                              |
| 00033          | DONE.                                                                          | P00047        |                              |                                                              |
|                |                                                                                |               | REF BY                       | 00042                                                        |

Figure 14-5. American National Standard COPY WITH COMDK Option

COBOL  
ALT #

S O U R C E L I S T I N G

REF C O M P I L E R C O M M E N T S  
LINE #

00034  
00035

STOP RUN.  
END PROGRAM.

00048  
00049

\*\*\*\*\* THE ABOVE LISTING CONTAINS 000 ERROR MESSAGES \*\*\*\*\*

\*\*\* THE ABOVE LISTING CONTAINS 001 WARNING MESSAGES \*\*\*

\* THE ABOVE LISTING CONTAINS 000 EFFICIENCY MESSAGES \*

COMPILATION TIME (MIN): ELAP CLOCK= 000.21 PROC= 000.03

00000 OVERFLOW READS 00000 OVERFLOW WRITES 22445 WORDS MEMORY USED 00002 LINKS USED ON \*3 FILE

14-15

DD26

Figure 14-5. American National Standard COPY WITH COMDK Option (cont.)



| COBOL<br>ALT # | SOURCE LISTING                                                                 | REF<br>LINE # | COMPILER COMMENTS                                               |
|----------------|--------------------------------------------------------------------------------|---------------|-----------------------------------------------------------------|
| 00001          | IDENTIFICATION DIVISION.                                                       | 00001         |                                                                 |
| 00002          | PROGRAM-ID. ANSCP2.                                                            | 00002         |                                                                 |
| 00003          | ENVIRONMENT DIVISION.                                                          | 00003         |                                                                 |
| 00004          | CONFIGURATION SECTION.                                                         | 00004         |                                                                 |
| 00005          | SPECIAL-NAMES.                                                                 | 00005         |                                                                 |
| 00006          | INPUT-OUTPUT SECTION.                                                          | 00006         |                                                                 |
| 00007          | FILE-CONTROL.                                                                  | 00007         |                                                                 |
| 00008          | SELECT MASTER-FILE ASSIGN TO A1.                                               | 00008         |                                                                 |
| 00009          | DATA DIVISION.                                                                 | 00009         |                                                                 |
|                |                                                                                | ***           | WA OBJECT-COMPUTER PARAGRAPH MISSING--<br>ASSUMED 6000 WITH EIS |
| 00010          | FILE SECTION.                                                                  | 00010         |                                                                 |
| 00011          | *EJECT                                                                         | 00011         |                                                                 |
| 00012          | FD MASTER-FILE LABEL RECORDS ARE STANDARD.                                     | 00012         |                                                                 |
| 00013          | 01 REC-1.                                                                      | 00013         | CONTAINS 60 CHARACTERS 10 WORDS                                 |
| 00014          | 02 NAME PIC X(38).                                                             | 00014         | STARTS IN CHARACTER POSITION 1                                  |
| 00015          | 02 FILLER PIC XX.                                                              | 00015         | STARTS IN CHARACTER POSITION 39                                 |
| 00016          | 02 ACCOUNT-NO PIC X(20).                                                       | 00016         | STARTS IN CHARACTER POSITION 41                                 |
|                |                                                                                |               | REF BY 00048                                                    |
| 00017          | WORKING-STORAGE SECTION.                                                       | 00017         |                                                                 |
| 00018          | 77 WORK-ACCOUNT-NO PIC X(20).                                                  | 00018         | REF BY 00046                                                    |
| 00019          | 01 WORK-REC-1.                                                                 | 00019         | CONTAINS 38 CHARACTERS 7 WORDS                                  |
| 00020          | *01 WORK-REC-1 COPY LIB-REC-1.                                                 | 00020         |                                                                 |
| 00021          | 03 ACCOUNT-NO PIC X(20).                                                       | 00021         | STARTS IN CHARACTER POSITION 1                                  |
|                |                                                                                |               | REF BY 00048                                                    |
| 00022          | 03 BALANCE PIC 9(16)V99 VALUE 0.                                               | 00022         | STARTS IN CHARACTER POSITION 21                                 |
| 00023          | 01 WORK-REC-1A.                                                                | 00023         | CONTAINS 76 CHARACTERS 13 WORDS                                 |
| 00024          | *01 WORK-REC-1A COPY LIB-REC-1.                                                | 00024         |                                                                 |
| 00025          | 03 ACCOUNT-NO PIC X(20).                                                       | 00025         | STARTS IN CHARACTER POSITION 1                                  |
|                |                                                                                |               | REF BY 00048                                                    |
| 00026          | 03 BALANCE PIC 9(16)V99 VALUE 0.                                               | 00026         | STARTS IN CHARACTER POSITION 21                                 |
| 00027          | 03 NAME PIC X(38) VALUE SPACES.                                                | 00027         | STARTS IN CHARACTER POSITION 39                                 |
| 00028          | 01 WORK-REC-2.                                                                 | 00028         | CONTAINS 44 CHARACTERS 8 WORDS                                  |
| 00029          | *01 WORK-REC-2 COPY LIB-REC-2.                                                 | 00029         |                                                                 |
| 00030          | 02 ACCOUNT-NO.                                                                 | 00030         | REF BY 00048                                                    |
| 00031          | 03 SUB-1 PIC X(10).                                                            | 00031         | STARTS IN CHARACTER POSITION 1                                  |
| 00032          | 03 SUB-2 PIC X(10).                                                            | 00032         | STARTS IN CHARACTER POSITION 11                                 |
| 00033          | 02 BALANCE PIC \$, \$\$\$, \$\$\$, \$\$\$, \$\$\$, \$\$\$, \$\$\$9.99 VALUE 0. | 00033         | STARTS IN CHARACTER POSITION 21                                 |
| 00034          | 01 WORK-REC-2A.                                                                | 00034         | CONTAINS 82 CHARACTERS 14 WORDS                                 |
| 00035          | *01 WORK-REC-2A COPY LIB-REC-2.                                                | 00035         |                                                                 |
| 00036          | 02 ACCOUNT-NO.                                                                 | 00036         |                                                                 |
| 00037          | 03 SUB-1 PIC X(10).                                                            | 00037         | STARTS IN CHARACTER POSITION 1                                  |
| 00038          | 03 SUB-2 PIC X(10).                                                            | 00038         | STARTS IN CHARACTER POSITION 11                                 |
| 00039          | 02 BALANCE PIC \$, \$\$\$, \$\$\$, \$\$\$, \$\$\$, \$\$\$, \$\$\$9.99 VALUE 0. | 00039         | STARTS IN CHARACTER POSITION 21                                 |
| 00040          | 02 NAME PIC X(38) VALUE SPACES.                                                | 00040         | STARTS IN CHARACTER POSITION 45                                 |
| 00041          | PROCEDURE DIVISION.                                                            | 00041         |                                                                 |
| 00042          | START SECTION.                                                                 | S00042        |                                                                 |
| 00043          | PARA-1.                                                                        | P00043        |                                                                 |
| 00044          | *PARA-1. COPY LIB-PARA.                                                        | 00044         |                                                                 |
| 00045          | OPEN INPUT MASTER-FILE.                                                        | C00045        |                                                                 |
| 00046          | MOVE SPACES TO WORK-ACCOUNT-NO.                                                | C00046        | REF TO 00018                                                    |
| 00047          | PARA-2. READ MASTER-FILE AT END GO TO DONE.                                    | P00047        |                                                                 |
| 00048          | MOVE ACCOUNT-NO OF REC-1 TO                                                    | 00048         | REF TO 00052<br>REF TO 00016 00021 00025 00030                  |

Figure 14-6. American National Standard COPY WITH CCOMDK Option

| COBOL<br>ALT # | SOURCE LISTING            | REF<br>LINE # | COMPILER | COMMENTS |
|----------------|---------------------------|---------------|----------|----------|
| 00049          | ACCOUNT-NO OF WORK-REC-1  | 00049         |          |          |
| 00050          | ACCOUNT-NO OF WORK-REC-1A | 00050         |          |          |
| 00051          | ACCOUNT-NO OF WORK-REC-2. | 00051         |          |          |
| 00052          | DONE.                     | P00052        |          |          |
| 00053          | STOP RUN.                 |               | REF BY   | 00047    |
| 00054          | END PROGRAM.              | 00053         |          |          |
|                |                           | 00054         |          |          |

\*\*\*\*\* THE ABOVE LISTING CONTAINS 000 ERROR MESSAGES \*\*\*\*\*

\*\*\* THE ABOVE LISTING CONTAINS 001 WARNING MESSAGES \*\*\*

\* THE ABOVE LISTING CONTAINS 000 EFFICIENCY MESSAGES \*

COMPILATION TIME (MIN): ELAP CLOCK= 000.21 PROC= 000.03

0000 OVERFLOW READS 0000 OVERFLOW WRITES 22445 WORDS MEMORY USED 00002 LINKS USED ON \*3 FILE

14-17

DD26

Figure 14-6. American National Standard COPY WITH CCOMDK Option (cont.)



## SECTION XV

### SEGMENTATION AND MODULARIZATION

#### TERMINOLOGY

When a standard concept of segmentation was extended to Series 60/6000 COBOL, it became necessary to revise terminology. Previously, the terms 'segmentation' and 'segments' were used to denote capabilities associated with the operating system. Where these words are used now, they will apply only to the standard COBOL concept of segmentation as discussed in this section. The term modularization will replace the previous usage of the term segmentation and the term module will replace the term segment.

Modularization, then, is the facility of combining separately compiled programs (modules) as building blocks to the solution of a problem using a non-COBOL CALL statement to effect serial transfer of control. This concept is a Series 60/6000 feature and therefore must be considered nonstandard.

Occurrences of the term 'segment-number' will be replaced by the term 'priority-number' in the COBOL Reference Manual and COBOL User's Guide in conformance to American National Standard COBOL-1968. However, the term 'segment-number' is utilized in the CODASYL COBOL Journal of Development and in American National Standard COBOL-1974. These terms may be considered interchangeable in Series 60/6000 COBOL.

#### DESCRIPTION OF SEGMENTATION

COBOL segmentation is a facility that allows the user to communicate with the compiler to specify object program overlay requirements.

Segmentation is concerned 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. (See Structure of Program Segments.) Segmentation in no way affects the need for qualification of procedure-names to ensure uniqueness.

## Segments

The segmentation feature permits the user to physically subdivide the Procedure Division of a COBOL object program. All source paragraphs that contain the same priority-number in their section headers are considered to be one segment at object program execution. Since priority-numbers can range from 00 through 99, it is possible to subdivide any object program into a maximum of 100 segments.

Program segments may be of three types, fixed permanent, fixed overlayable, and independent.

Fixed permanent segments are always in memory during the execution of the entire program; i.e., they cannot be overlayed except when the system is executing another program, in which case fixed segments may be 'rolled out' temporarily.

Fixed overlayable segments may be overlayed during program execution, but any such overlaying is transparent; that is, fixed overlayable segments are logically identical to, but physically different from, fixed segments. A fixed overlayable segment, if called for by the program, is always made available in its last used state.

Independent segments may be overlayed, but such overlaying will result in the initialization of that segment. Therefore, independent segments are logically different from fixed permanent/fixed overlayable segments, and physically different from fixed segments. Independent segments are numbered 50 through 99.

The logical/physical characteristics of the three types of segments are shown below, where L indicates that the segment types are logically identical, and P indicates that they are physically identical.

|                   | Fixed Permanent | Fixed Overlayable | Independent |
|-------------------|-----------------|-------------------|-------------|
| Fixed Permanent   | -               | L                 | None        |
| Fixed Overlayable | L               | -                 | P           |
| Independent       | None            | P                 | -           |

## Segment Classification

Sections which are to be segmented are classified, using a system of priority-numbers included with Procedure Division section headers, and the following criteria:

1. 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 fixed permanent segments; sections which are used less frequently are normally classified as belonging either to one of the fixed overlayable segments or to one of the independent segments, depending on logic requirements.

2. 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.
3. 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. If the priority-numbers are not in strict ascending order, a prepass must be made by the compiler. This procedure is activated when the SEGMNT option is included in the variable field on the \$ COBOL card. (Refer to the segmentation deck setup example in Appendix B.)

The SEGMNT option should not be used when sections are already in ascending priority-number order.

If the segmented source program is an I-D-S program, the SEGMNT option must be included in the variable field on the \$ IDS card.

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.

### Structure of Program Segments

#### PRIORITY-NUMBERS

Section classification is accomplished by means of a system of priority-numbers. The priority-number is included in the section header:

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-number is assumed to be 0.

Sections in the declarative portion must contain priority-numbers having a value of 0 only.

All sections which have the same priority-number constitute a program segment.

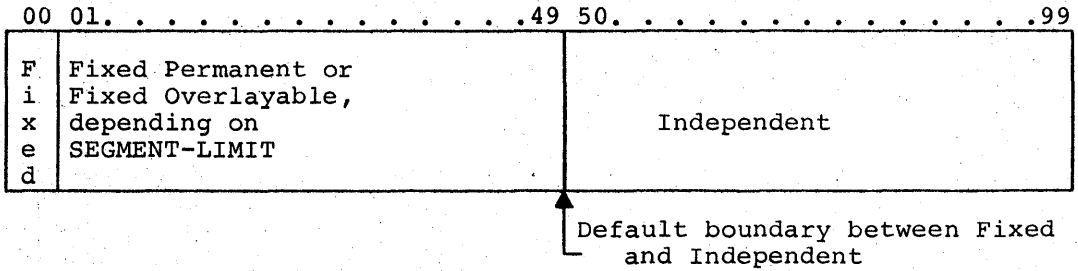
Segments with priority-number 0 through 49 belong to the fixed portion of the object program.

Segments with priority-number 50 through 99 are independent segments.

**SEGMENT-LIMIT**

Unless overridden by the user, all segments numbered 00 through 49 are fixed permanent segments, and segments numbered 50 through 99 are independent segments.

If, however, the user requires fixed overlayable segments, they are numbered from a user-specified value (01 through 49). The user specifies the lowest numbered segment which is to be fixed overlayable in the SEGMENT-LIMIT phrase of the Environment Division. Therefore, the more fixed overlayable segments there are, the fewer fixed permanent segments there can be. Segment 00 is always fixed. The relationship between segment types and priority-numbers is shown below.



The logical relationship between all segments numbered 00 through 49 is always the same, regardless of SEGMENT-LIMIT. For example, an altered GO TO statement which appears in a segment numbered 27 will remain altered, whether the segment is fixed permanent or fixed overlayable, until the execution of another ALTER statement; intervening overlays of the segment will not result in initialization of the segment. Therefore, COBOL paragraphs numbered 00 through 49 can be written as if all such paragraphs were always fixed in memory, and subsequent changes in SEGMENT-LIMIT will have no effect on program logic.

The SEGMENT-LIMIT phrase appears in the OBJECT-COMPUTER paragraph 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 phrase 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 fixed overlayable segments.

When the SEGMENT-LIMIT phrase is omitted, all segments having priority-numbers from 0 through 49 are considered as permanent segments of the object program.

## Transfer of Control

Four methods are used to transfer control within a program:

1. A GO TO statement.
2. A PERFORM statement.
3. An input procedure or output procedure associated with a SORT statement.
4. An output procedure associated with a MERGE statement.

An input procedure or output procedure can be considered an implicit PERFORM which is executed in conjunction with a SORT or MERGE statement. For that reason, the restrictions on the PERFORM statement apply equally to input procedures and output procedures.

The CALL statement transfers control from a calling program to a called program and is not, therefore, relevant to a discussion of segmented programs.

The ALTER statement, while not in itself transferring control, affects control transfer by altering a future control path. Its effect on transfer of control is included in this section for that reason.

### Restrictions on Transfer of Control and Program Alteration

The segmentation feature imposes no restrictions on transfer of control as long as the control path remains within the range of fixed permanent and fixed overlayable segments. These kinds of segments are logically identical and can be treated by the user as though the program were not segmented. If, however, independent segments are involved in the transfer of control, some restrictions exist due to the transient nature of independent segments.

The restrictions imposed by segmentation occur only under the following circumstances:

1. When a PERFORM, ALTER, SORT, or MERGE statement that is not in an independent segment refers to a paragraph, input procedure, or output procedure that is in an independent segment.
2. When a PERFORM, ALTER, SORT, or MERGE statement in an independent segment refers to a paragraph, input procedure, or output procedure in a different independent segment.
3. When a PERFORM, SORT, or MERGE statement in an independent segment refers to a paragraph, input procedure, or output procedure that is not in an independent segment, which in turn refers to an independent segment.

Segmentation imposes no specific restrictions on GO TO statements, but it does restrict the use of GO TO statements which transfer control in violation of the restrictions on PERFORM, ALTER, MERGE, or SORT statements.



Restrictions are required since independent segments are loaded into memory under control of an operating system over which the user has no direct control. (The user can never predict when any given independent segment will be loaded.) Since memory may only be sufficient to contain one independent segment at a time, the user must not execute statements that depend upon the presence, in memory, of any given independent segment (other than the one in which the statements appear) at any given time.

The specific restrictions imposed on the PERFORM statement by segmentation are:

- 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:
  - a. Sections, each of which has a priority-number less than 50.
  - b. 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:
  - a. Sections, each of which has the same priority-number as that containing the PERFORM statement.
  - b. Sections with a priority-number that is less than the SEGMENT-LIMIT.

One restriction is imposed on the ALTER statement by segmentation. Since the presence of independent segments in memory is unpredictable, paragraphs whose priority-number is greater than or equal to 50 must not be referred to by an ALTER statement in a section with a different priority-number.

The restrictions on the SORT or MERGE statements are identical to those imposed on the PERFORM statement, because a SORT or MERGE statement is considered to be an implied PERFORM of associated input procedures or output procedures.

### Segmentation, Linking, and Loading

COBOL segmented programs are loaded by the General Loader program.

Because of program size and complexity, it may be necessary to segment a program to make more efficient use of memory and available storage media. Each of these segments may be referred to as a 'link'. When using links, the program must be organized in such a way to retain the more commonly used subprograms in the links that will reside in memory and the lesser used subprograms in links that will be used as temporary overlays. All of the subprograms loaded that precede the first \$ LINK card and all subprograms loaded as a result of the first library search are commonly referred to as being in the 'main link'.

The \$ LINK card is used to specify the positions in the output stream at which segmentation is to take place. When this card is encountered, all requested library files are searched to satisfy any undefined references (SYMREFs) in the link being terminated. The \$ LINK card specifies, in its first variable field, a unique identifier for the new link. When variable field 2 is present on the card, it indicates that the new link is to overlay a previously loaded link(s) whose identifier appears in field 2. In this condition, the new link assumes the origin of the link specified in field 2. All links which are to be overlaid by the new link are written in system loadable format onto the file that has the file-code H\*.

The amount of cross-reference between the various subprograms that make up the link program dictates the desired segmentation. The main link, as defined above, should contain all high-usage subprograms because of its permanent status in memory during program execution. Subprograms contained in any other link are always able to reference subprograms in the main link.

COBOL segmented decks are identified by the numeral one (1) in bit 20 of the third word of the preface card. Special preface cards precede each segment in the object deck and are similar in function to \$ LINK cards.

A COBOL segmented deck is loaded as described below:

1. When a segmented deck is detected, a test is made to determine if the \$ OPTION COBOL card is included. If this card is not present, loading is terminated.
2. The symbols from the preface card are loaded into the load table. The preface information is then saved in a pushdown stack.
3. Required library routines are loaded.
4. The saved preface information is restored to continue loading the segmented program. Labeled Common storage areas are assigned, followed by space for the program.
5. The coding that precedes the first special preface card is loaded. This coding, together with constants and working-storage, forms the fixed portion of the program that always resides in memory.
6. Load parameters from the special preface card are saved and the coding that follows is loaded until another preface card is detected. The instructions forming the current segment are written on the H\* file with MME GESAVE.

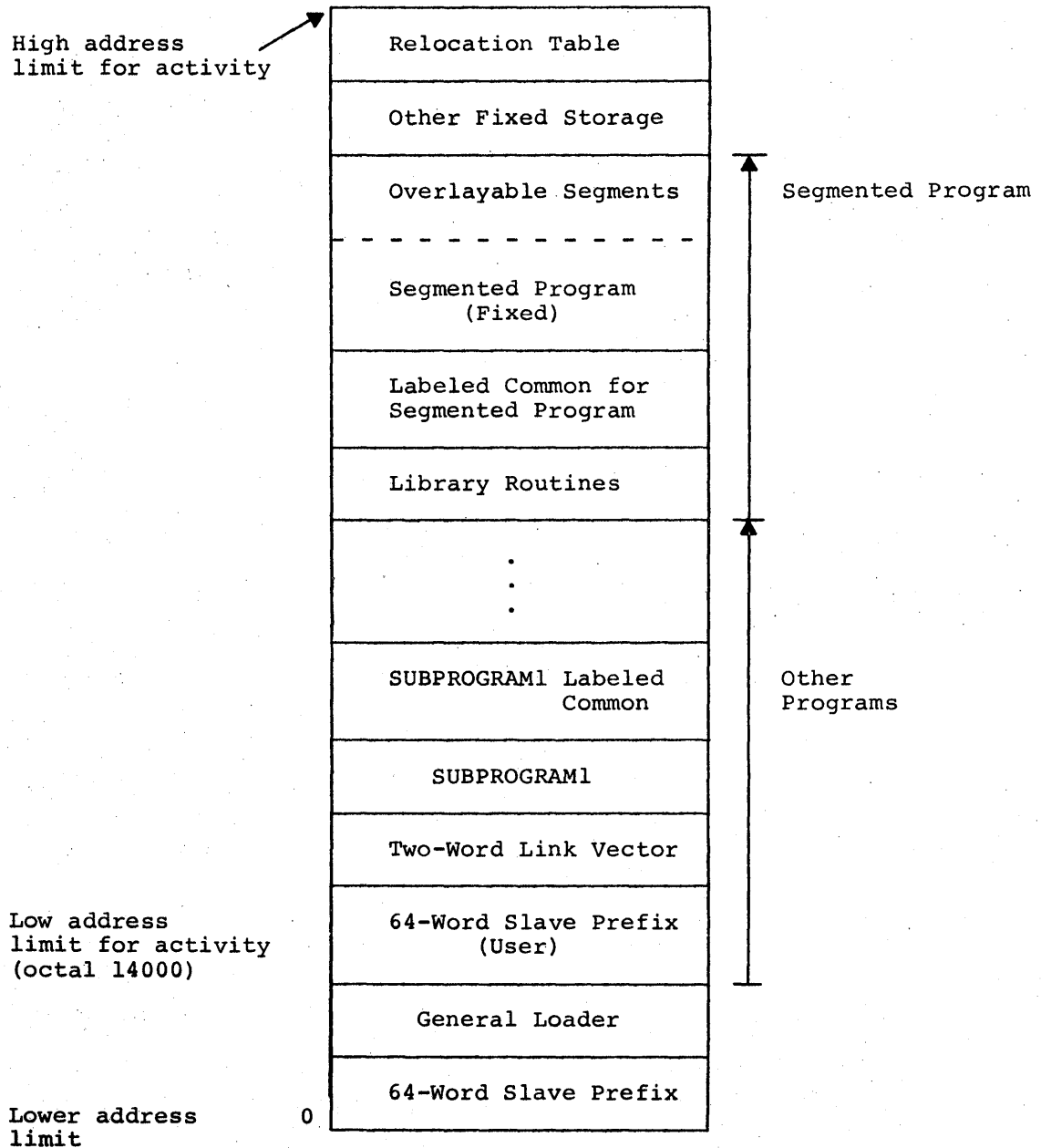
This process is repeated until the special preface card that identifies the last segment is detected. The last segment is written on the H\* file.

7. The remaining coding that is part of the fixed permanent portion is loaded. When a \$ EXECUTE or \$ LINK card is encountered, the fixed portion is written on the H\* file.

The manner in which segmented programs are loaded requires the user to follow the conventions listed below:

1. A \$ OPTION COBOL card must precede the first segmented program.
2. If present, the following cards must precede the segmented program:
  - \$ LIBRARY
  - \$ NLOAD
  - \$ NOLIB
3. If several object decks are present in the load, and at least one of the segmented overlays references another object program through a SYMREF/SYMDEF relationship, the object program containing the SYMDEF must precede the segmented program.
4. If more than one segmented program is present, communication is possible only between the fixed permanent portions.
5. If \$ LINK cards are present, communication is possible only between the permanent portion of each link.
6. Using \$ EQUATE cards to define new SYMDEFs is restricted to SYMDEFs that are already defined.
7. The variable field of the \$ EXECUTE card must contain the SEG option if the user desires allocation of H\* by GCOS. If the SEG option is not present, a peripheral control card should be provided for the H\* file.
8. If neither of the conventions specified in item 7 are followed, the General Loader attempts to create an H\* file via GEMORE at the first GESAVE. If the GEMORE is denied, the General Loader places an error message on the P\* file and aborts.

The following diagram illustrates the memory layout for a segmented program with additional subprograms.



For additional information, refer to the General Loader Reference Manual.

Effects of Segmentation on Listings

The listing of cross-references between permanent segment procedures and overlayable segment procedures can be made indirectly rather than directly by using a source card line number. The indirection involves a table in the GMAP listing called the Procedure-Names Table which is located by the symbol '.CTABL'. The following example describes the relationship between cross-references for a segmented program with an independent segment (priority-number = 52). Note that the procedure with line number P00105 has no REF BY comment. Instead, its cross-reference is printed at line number 00007 which corresponds to the seventh entry in the Procedure-Names Table of the GMAP listing. The GO TO statement at line number 00154 also has a REF TO comment that points to line number 00007 (an indirect reference).

Example:

|                                                | REF<br>LINE # | COMPILER COMMENTS |
|------------------------------------------------|---------------|-------------------|
| REMARKS. THIS IS A SEGMENTED PROGRAM.          | 00007         | REF BY 00154      |
| .                                              |               |                   |
| START SECTION.                                 |               |                   |
| PARA-A. READ IN-FILE AT END GO TO DONE. P00105 |               |                   |
| .                                              |               |                   |
| ERR SECTION 52.                                |               |                   |
| (This is an independent segment)               |               |                   |
| .                                              |               |                   |
| GO TO PARA-A.                                  | 00154         | REF TO 00007      |

Then, in GMAP:

PROCEDURE-NAMES TABLE

| TTLS     | PROCEDURE-NAMES | TABLE  |                   |
|----------|-----------------|--------|-------------------|
| .CTABLBS | 000000          |        |                   |
| BCI      | 01,S00092       | 000001 |                   |
| ZERO     | 000000,S00092   |        |                   |
| BCI      | 01,000001       | 000002 |                   |
| ZERO     | 000000,P00094   |        |                   |
| BCI      | 01,000001       | 000003 |                   |
| ZERO     | 000000,S00095   |        |                   |
| BCI      | 01,000001       | 000004 |                   |
| ZERO     | 000000,P00097   |        |                   |
| BCI      | 01,000001       | 000005 |                   |
| ZERO     | 000000,S00100   |        |                   |
| BCI      | 01,000001       | 000006 |                   |
| ZERO     | 000000,P00101   |        |                   |
| BCI      | 01,000001       | 000007 | ← entry<br>number |
| ZERO     | 000000,P00105   |        |                   |

## Summary of Segmentation Requirements

The main features of segmentation are listed below:

1. COBOL segmentation is concerned only with the segmentation of procedures in the Procedure Division.
2. When used, the entire Procedure Division must be in sections.
3. Section classification is accomplished by means of a set of priority-numbers ranging in value from 0 through 99 in the section header format:

section-name SECTION [ priority-number ] .

4. When a priority-number is omitted from a section header, it is assumed to be zero, as are sections in the declarative portion.
5. Sections need not be divided physically into ascending logical order of priority-number; however, such ordering is preferable since compile times will be increased to rearrange the source program. (This is the case when the SEGMNT option on the \$ COBOL card is required.)
6. The SEGMENT-LIMIT feature provides a means by which the number of permanent segments of a program can be reduced from 49 to 1. This phrase is used in the OBJECT-COMPUTER paragraph with the format:  

[ SEGMENT-LIMIT .IS priority-number ]
7. When the SEGMENT-LIMIT phrase is omitted, all segments having priority-numbers from 0 through 49 are considered as permanent segments of the object program.
8. When the SEGMENT-LIMIT phrase 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.

## DESCRIPTION OF MODULARIZATION

The objectives of COBOL program modularization are:

1. To permit practical separation of a data processing program into distinct functional components (modules).
2. To permit the modules to be developed as separate COBOL source programs that are compiled separately and may be debugged separately.
3. To permit programs to be linked by the object program loader.
4. To permit the functional modules to overlay other modules when called into memory in order to execute large programs within a limited amount of memory.

Three distinct communication problems arise in modularizing data processing programs. The first is the communication of information contained in the data file buffers and housekeeping information that is common between two or more modules. The second is the communication of working-storage data common to two or more modules. The third is the communication of procedural control. Other problems arise when a data processing program is modularized to function in an overlay environment. One problem that must be considered when operating in an overlay environment is how to control files which are common to the two or more modules. There should be a communication capability that allows loading of an overlay module which restores any common areas to their initial states or allows them to remain in their current states.

As a special feature, the COBOL compiler provides solutions to all three communication problems, as described below. It should be understood that source program modularization (as specified for COBOL) must generally be rearranged into a unified, nonmodularized form if it is to be subsequently compiled on a different computer. Such rearrangement is not necessary for source programs that do not use the modularization feature.

### Modules

Modules are separate programs, compiled and tested independently, and subsequently loaded together and executed as a total program. In this manner, a large complex program may be divided into several parts (modules), each part written as a separate source program, and each part compiled and tested independently, thereby overlapping programming and checkout time. Another use of modules is to facilitate writing common subroutines (installation oriented) in source language to be compiled as independent modules.

### Sections

Sections consist of a section header followed by zero, one, or more successive paragraphs. They generally contain a common function that is executed from more than one location in a program. Any program may be partitioned into sections. A section ends immediately before the next section-name or at the end of the Procedure Division or, in the declarative portion of the Procedure Division, at the keywords END DECLARATIVES.

The compiler provides segmentation of COBOL procedural sections as specified in American National Standard COBOL specifications. However, through modularization, it is possible to organize a group of COBOL modules functionally to operate in the same manner as segmentation of American National Standard COBOL-1968 procedural statements, with the exception that there is no way to call a segment and have it made available in its last used state.

An explanation of the three methods of communication between modules follows:

1. The COBOL compiler automatically places the file control blocks and all buffers for each file named in the File Section into Labeled Common storage. The name of the Labeled Common storage area is assigned using the two-character file-code specified in the ASSIGN phrase for the file. At load time, the loader will allocate all Labeled Common storage areas having the same name to the same area of memory. Therefore, if a file is referenced by two or more modules, it must be described identically in the Data and Environment Division of all source programs referencing it. Identical file properties include such clauses as VALUE OF IDENTIFICATION IS literal-1 in the FD entry, since these clauses affect the length of Labeled Common storage for a file. There may be times when it is desired to allocate the file buffers to Blank Common storage rather than to Labeled Common storage. This can be accomplished by using the BLANK COMMON phrase in the FILE-CONTROL paragraph. This can be very useful if it is desired to allow the loader to share the buffer areas of the file at load time to decrease memory requirements for job allocation. For details on loading the Blank Common storage area versus the Labeled Common storage area, refer to the General Loader manual.
2. A report can be referenced only from within the module in which it is described. If several reports going to a single file are to be generated in separate modules, each report must be specified in the module which contains the relevant Report Writer verbs, and may be omitted from modules not containing relevant Report Writer verbs. The complete file description, excluding unreferenced report descriptions, must appear identically in each reporting module.
3. Files sharing the SAME AREA or SAME RECORD AREA must all be described in any module referencing any of them. Files common to two or more modules may be defined to share buffer areas with other files common to the same modules. However, when indicating this, the SAME AREA phrase must be identical for all modules that share the common files.

A source program may include as many BLOCK clauses as needed. The sum of the number of file-codes and BLOCK labels must not exceed 63 for a given module.

#### Procedure Division Communications

A PROGRAM-ID paragraph must be specified for each source program. The PROGRAM-ID designator must be one to six characters in size and must be composed of letters and/or digits, including at least one letter.

The compiler uses the PROGRAM-ID program-name for the implied entry symbol for identifying the COBOL object program. The implied entry point for a source program is the first procedural statement following the END DECLARATIVES statement if declaratives are used, or the first statement of the Procedure Division if declaratives are not used. The entry symbol (the PROGRAM-ID) is made a global symbol which allows referencing by calls from other modules.



Each COBOL object program has the basic structure of a single-entry closed subroutine. This is true even though any number of entry points may be defined within each module. The compiler automatically generates the entry linkage coding at the entry point (the PROGRAM-ID), and the implied exit linkage coding is generated at each entry point and exit point to save and restore all index registers when an entry is called as a closed subroutine. Inter-communication between COBOL object program modules does not require index register integrity to be maintained. The Save and Restore index register feature is implemented in COBOL to allow communication between COBOL modules and non-COBOL modules when operating in a modularized environment. If a module has only one entry point (the implied entry), and it is to be executed as a closed subroutine by another module, the program should be arranged to 'fall through' to the exit linkage at the appropriate time. An EXIT statement may be used for this purpose if necessary.

## CALL STATEMENT

The CALL statement may be used to transfer control to a separately compiled program or to an entry point within a program, with a standard return mechanism provided. An independently compiled COBOL program may be called as a closed subroutine by a CALL statement, using the PROGRAM-ID that names the module. If only a portion of a program taken from another program is required to be executed, the CALL statement should reference one of the entry-names defined in an ENTRY POINT phrase (Format 4 of the ENTER statement). A CALL statement can reference non-COBOL subprograms if the called routine-name has been established as a global symbol within the called module.

It should be emphasized that the CALL statement provides only a means of transfer of control and does not cause a program to be loaded from an H\* file for overlay environments. If a program is submitted to the operating system together with a \$ LINK control card, it must be loaded by CALL LLINK coding in a COBOL source statement before it can be called using the PROGRAM-ID program-name.

If a COBOL program has a PROGRAM-ID named LINKA, and it has been loaded with a \$ LINK ALINK control card, control may be assigned as follows:

```
01 L1 PIC X(6) VALUE "ALINK".
.
.
CALL LLINK USING L1.
CALL LINKA.
```

For each CALL statement, USING arguments can be specified to provide address pointers to data that is to be used by the entry-name being called. The USING arguments are not meaningful if the CALL statement references the PROGRAM-ID of a COBOL subprogram. The arguments provide indirect pointers to input and output data fields when a CALL statement references an entry-name that is defined using the ENTRY POINT phrase and includes USING and GIVING arguments. Data-names specified as USING arguments must be level 77 or 01 data items defined in the Working-Storage Section of the called subprogram.

The number of USING arguments specified with a CALL statement must correspond exactly with the number of USING and GIVING arguments defined for the entry-name that is referenced by the CALL statement. The data descriptions for each of the corresponding arguments must be identical when a CALL statement references a COBOL program. If the CALL statement references a non-COBOL program, the data descriptions should provide a data format that is compatible with the called program.

#### ENTRY POINT PHRASE

The ENTRY POINT feature is provided to define entry points that are in addition to the implied entry point. Using this facility, a program may be organized so that paragraphs, sections, or combinations of paragraphs and sections can be called and executed from other modules.

Each entry-name must be unique and must not contain more than six characters, with at least the first two characters letters and the remaining characters defined as letters and/or digits. An ENTRY POINT phrase may be used in any location in the Procedure Division except in the declarative portion. Entry-names can be referenced only through calls from other modules. They must not be referenced by a CALL statement from within the module in which they reside. The compiler generates one nine-word 'save area' in each module for the preservation of index registers and indicators. Therefore, a segment which has been called may itself contain CALL statements. However, a called module or entry point must not contain a CALL statement that directly or indirectly calls the calling program.

#### EXIT STATEMENT

The compiler provides an EXIT entry-name option that defines an unconditional exit for a given entry-name. Each entry-name (explicit) in a module must have at least one exit defined using the EXIT entry-name statement. Multiple exits may be defined for an entry-name if necessary. If control reaches an EXIT entry-name statement, the linkage control stack is checked to determine if the current EXIT corresponds to the called entry-name. If the EXIT corresponds to the entry-name, the EXIT causes control to return to the calling program immediately following the CALL statement. If it does not correspond to the entry-name, control passes through the exit point to the first sentence of the next paragraph. An entry-name with its associated EXIT and/or EXITS can be nested within other entry-names and their associated EXITS. ENTRY POINTS and EXITS may be placed so that they extend across other entry-names and/or exits. Since the same pushdown stack is used for entry-names that are used to process paragraphs, orderly pushdown and popup of the control stack should be provided when processing paragraphs within coding for an entry-name. The EXIT PROGRAM option is implemented to provide an unconditional exit from a module when operating under the control of a CALL statement. This feature causes control to return 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 passes through the exit point to the first sentence of the next paragraph.

## Data Compatibility

COBOL provides excellent facilities for processing common data between program modules. Since all files are assigned to Labeled Common storage areas, considerable flexibility is provided for referencing common data when only COBOL modules are involved.

An additional feature is available to allow the communication of data between non-COBOL modules and COBOL entry points, other than the implied entry point, by means of arguments. USING and GIVING arguments may be specified with the ENTRY POINT phrase. Any USING arguments associated with an entry-name reference data items within the module; these data items function as receiving fields for moving input arguments that takes place when the entry-name is called from another module. The compiler assumes that the external format of the item is compatible with the data description specified for the item in the Data Division. The order and descriptions of the arguments must conform to the calling program's USING argument list. No more than ten USING and GIVING arguments may be defined with an ENTRY POINT phrase. USING argument data-names must reference level 77 or 01 data items described in the Working-Storage Section. The Procedure Division statements that follow an ENTRY POINT phrase are not executed until individual moves of the USING argument list have been completed.

The GIVING arguments specified with an entry-name reference data items in the Working-Storage Section that function as sending fields for moves of output arguments which take place when an EXIT entry-name statement is encountered. The compiler assumes that the desired external format is compatible with the data description specified for the item in the Data Division. GIVING argument data-names must reference level 77 or 01 data items described in the Working-Storage Section. Control is not returned to a calling program when a valid EXIT statement is encountered until individual moves of the GIVING argument list have been completed. The GIVING moves use the corresponding argument address associated with the controlling CALL statement from the calling program as a receiving field. A one-to-one correspondence must exist between the CALL...USING arguments and the ENTRY POINT USING/GIVING arguments.

Data item format descriptions in COBOL that might be compatible with external formats of non-COBOL modules are DISPLAY, COMPUTATIONAL-1, COMPUTATIONAL-2, and COMPUTATIONAL-3. The COMPUTATIONAL format should not be used to describe USING or GIVING data items. If it is desired to maintain a high degree of decimal precision when performing computations involving USING data-names, the required conversion can be accomplished by moving the data item to a field defined appropriately as COMPUTATIONAL.

## File Compatibility

If more than one COBOL program in a run unit defines a file using the same file-code, the following restrictions apply:

1. The file definitions must be identical.
2. All but one of the file definitions must specify the OVERLAY phrase in the SELECT sentence of the FILE-CONTROL paragraph. This restriction applies even if no overlays are contained in the run unit.

## LINKED OVERLAY ENVIRONMENT CONSTRAINTS

A run unit employing linked overlays must be carefully organized and constructed in accordance with the "Link/Overlay Processing" discussion in the General Loader manual. The following factors may require special consideration in a linked overlay environment.

### File Processing

The opening of a file causes the insertion of the associated file control structure into a circular threaded list of open files. The closing of a file causes the removal of the associated file control structure from the open file list, and the relinking of that list to maintain its integrity. Both of these actions involve the manipulation of address pointers to and within the file control structures in the open file list in addition to those associated with the file(s) being opened or closed.

Overlaying any active file control information (for any open files) may destroy the continuity of the open file list and cause unpredictable behavior of any input-output activity that may occur while such a discontinuity exists.

In certain instances, the file control structures for open files contain address pointers to object program or subroutine procedures (for label or error processing declaratives, for example) which may be engaged in response to external events or other program procedural interactions. The procedures referenced by such pointers remain active as long as the associated file remains open.

Overlaying active input-output procedures, whether contained in object programs or system subroutines, may result in the indeterminate behavior of any asynchronous or event-driven references to such procedures occurring while they are not present in their established memory locations.

### ACCEPT and DISPLAY Statements

ACCEPT and DISPLAY statements that reference system input (GIN) or system output (SYSOUT) both involve the use of implicit file processing mechanisms and control structures, and thus inherit the file processing considerations discussed above. Additional limitations upon the use of ACCEPT and DISPLAY statements in a linked overlay environment result from the lack of any explicit source program syntax for the opening or closing of the implicit files associated with those statements, or for the recognition of the end-of-file condition on the system input file.

The first execution of a DISPLAY statement referencing the system output facility causes automatic opening of the implicit system output file, thereby inserting its file control structure into the open file list. Once opened, the system output file remains open for the duration of the run unit, and is closed only by the wrapup process during the execution of the STOP RUN statement.

Since the file control structure for the system output file resides in the subroutine that services the associated DISPLAY statement, that subroutine cannot be safely overlaid after the first execution of a DISPLAY statement in a run unit.

The first execution of an ACCEPT statement referencing the system input facility causes automatic opening of the implicit system input file, thereby inserting its file control structure into the open file list. Once opened, the system input file remains open until the end-of-file condition is sensed on that file, at which time it is automatically closed and removed from the open file list.

Since the file control structure for the system input file resides in the subroutine that services the associated ACCEPT statement, that subroutine cannot be safely overlaid while the file is open. In fact, lacking explicit COBOL syntax for the recognition of the end-of-file condition or closure of the system input file, the best alternative is to ensure that the subroutine that services the associated ACCEPT statements is not overlaid at any time after the first execution of an ACCEPT statement in a run unit.

### Overlay Management and Memory Organization

Whenever the overlay management strategy employed in a run unit can result in one of the foregoing situations, consideration must be given to reorganizing the run unit, either by rearranging the programs therein or by means of explicit General Loader directives.

File control structures may be repositioned to any suitable link by including a labeled common definition in the desired link with a \$ USE card:

```
$ USE fc/size/
```

where the symbol 'fc' is the file-code associated with the file (the first file-code appearing in the ASSIGN phrase of the SELECT sentence for the file), and 'size' is the number of words (in decimal) to be allocated for the labeled common area. The minimum size required may be determined from the load map or from the BLOCK LENGTH information appearing in the Preface of the assembly listing of the program.

User programs or subprograms may be placed in any suitable link in the loader input stream, while the rearrangement of any programs or subprograms obtained from user or system libraries may be accomplished by placing a reference thereto within the desired link, either with source program syntax or with a \$ USE card:

```
$ USE symbol
```

citing a primary SYMDEF in the desired module.

The ACCEPT and DISPLAY (system input and system output) subroutines and their file control structures may be forced into any suitable link by including ACCEPT or DISPLAY statement(s) as appropriate in some COBOL program in the desired link (the statement need not be executed), or by placing either or both of the following control cards in that link:

```
$ USE .CIGIN (for ACCEPT)
or $ USE .COSYS (for DISPLAY)
```

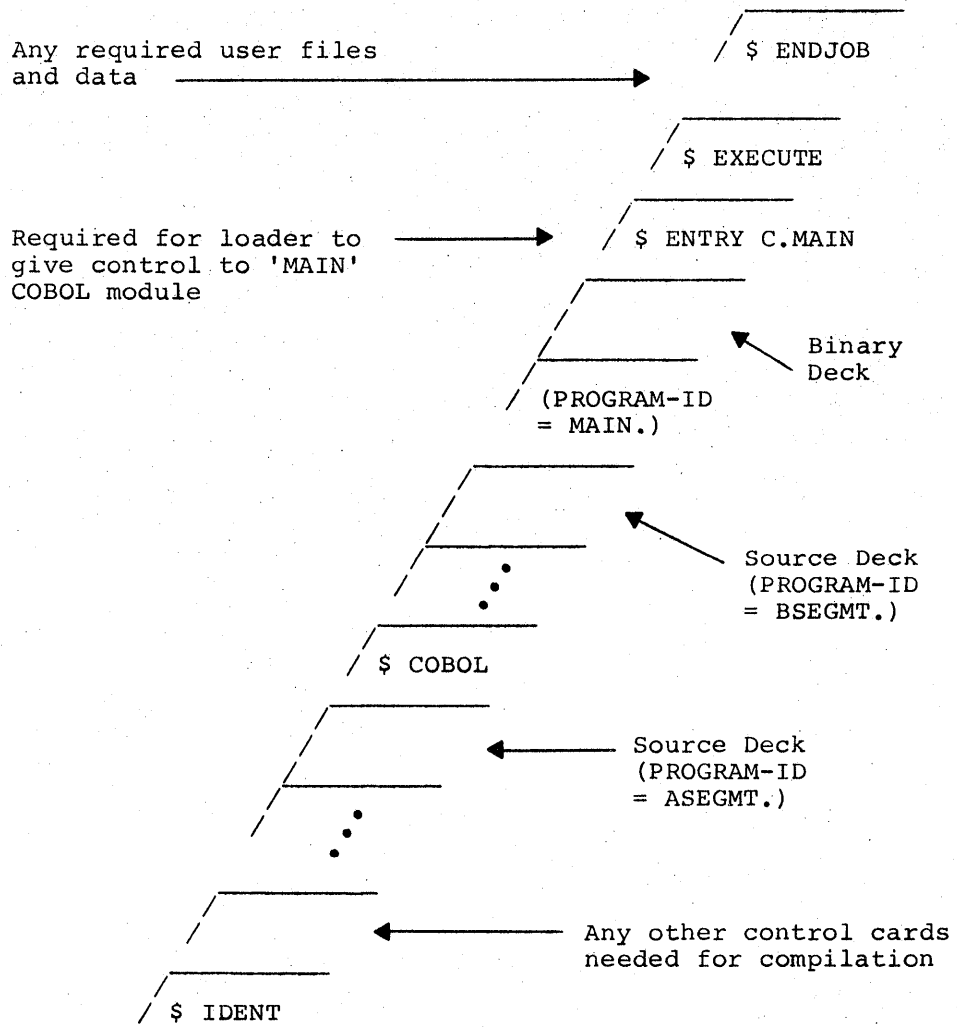
Whenever optional processing facilities for direct access or transliteration are to be included in a COBOL run unit, and that run unit is operating as a linked overlay structure, the required \$ USE control cards must be included in the definition of the main module rather than in any of the overlays. Any subsidiary modules, such as transliteration tables, must also be positioned in the main module.

Whenever direct-access or transliteration features are to be used in a run unit containing a segmented COBOL program, \$ USE control cards must be inserted into the job stack prior to the COBOL program.

### Multiple Module Program Example

An example of a three-module COBOL program follows. The example suggests that the MAIN module (PROGRAM-ID is MAIN) has already been compiled and that two other modules must be recompiled together with an execution activity of the three modules. Since the General Loader would give control to the first primary SYMDEF encountered in the deck setup, it is necessary in this example to use a \$ ENTRY card to force control to be given to the MAIN module. COBOL conventions provide a primary SYMDEF for this purpose. It is a six-character symbol in which the first two characters are always 'C.'; the remaining four characters are taken from the first four characters of the PROGRAM-ID program-name.

Deck Setup for Multiple Module Program:



## SECTION XVI

### EFFICIENCY TECHNIQUES

#### OPTIONAL DEBUG STATEMENTS

Standard source program statements may be selectively compiled or bypassed during the debugging phase of the compilation process. One of the digits 0 through 9 may be placed in column 7 of the source card on which the statement appears to indicate which statement(s) is to be compiled or bypassed. The compiler is then notified which statements are to be compiled or bypassed with an entry that is specified in the SPECIAL-NAMES paragraph of the Environment Division.

This efficiency technique saves checkout time by using COBOL source statements rather than memory or tape dumps to trace program errors. Some methods are:

1. Special counters can be used to record test information; the test results can be displayed online.
2. Intermediate results or the status of selected records before and after processing can be written to SYSOUT or to a special test file.
3. Literal values can be moved to data items and logical sequences or computational results checked.
4. The actual sequence of program execution can be traced by placing DISPLAY statements at strategic points.

By using these debugging features, much of the program checkout operation can be accomplished at the source language level.

Column 7 of the source statement card is used to identify those entries that may be compiled or bypassed. Any of the digits 0-9 in column 7 can be used to identify debugging statements. Normally, a particular digit is used to identify groups of debugging statements used for different purposes. To indicate to the compiler which debugging statements are to be compiled, one of the following options in the SPECIAL-NAMES paragraph must be specified:

- PROCESS ALL DEBUG STATEMENTS.
- PROCESS LEVEL integer-1 [ THRU integer-2 ] DEBUG STATEMENTS.



The first format is used when all of the debugging statements (those identified as such by a single digit in column 7) are to be compiled. The second format is used when either a single group of statements (all statements identified by the same digit) is to be compiled or when a range of debugging statements is to be compiled. When the checkout phase is completed, the DEBUG option should be removed from the SPECIAL-NAMES paragraph and the program should be recompiled to remove coding for debugging statements from the object program.

This feature may be used in the Input-Output Section of the Environment Division, in the Data Division, or in the Procedure Division, except in any entry containing a COPY clause or COPY statement.

Do not attempt to continue an optional compilation statement on a second line using a hyphen in column 7. These procedures are mutually exclusive.

### COMPILATION TECHNIQUES

The process of compilation is divided into two broad phases of activity. In phase 1, the source program is analyzed to determine lexical, syntactic, and semantic accuracy and is translated into an intermediate language. In phase 2, the intermediate language is converted to the appropriate machine language and assembled into an executable object program. Since the diagnostic functions, error messages, and cross-referenced source listings are prepared in phase 1, it is not necessary to continue compilation into phase 2 when the object program is not desired or when the object program produced from an erroneous source program would be useless.

If the object program is not desired, compilation time can be reduced by as much as 50 percent by specifying the following option in the SPECIAL-NAMES paragraph:

COMPILE PHASE1 ONLY [ WITH SOURCE ERRORS ] .

If the WITH SOURCE ERRORS phrase is specified, object coding will not be generated if fatal errors are detected when the source program is being analyzed. Warning and efficiency messages do not terminate the compilation. A source program listing with cross-references and messages is always produced. If no source program errors are detected, compilation continues and an executable object program is produced. If the WITH SOURCE ERRORS phrase is not specified, object program coding is not available under any circumstances.

The COMPILE PHASE1 ONLY option can be implied by specifying the COMPH1 option in the variable field on the \$ COBOL card. The effect of the COMPH1 option is the same as if the WITH SOURCE ERRORS phrase were omitted from the COMPILE PHASE1 ONLY option.

The NLSTIN option on the \$ COBOL card may also be used to reduce compilation time. If the NLSTIN option is included, the source program listing will be suppressed. However, any statement flagged with a fatal error message (\*\*\*\*\*) will be listed and the total number of fatal errors will be printed.

The amount of time required to compile a COBOL program is subject to considerable variation, depending upon the syntactic accuracy with which the program is coded and the resources allocated to the compilation. The following recommendations specify techniques for operating the Series 60/6000 COBOL compiler at a high level of performance. These recommendations are compiler/machine dependent and may not have the desired effect in other compiler/machine configurations.

### Unified Data Tables

The user can significantly affect the efficiency of the compilation in the Data Division of the source program. As the compiler processes Data Division entries, each entry is stored in a unified data table. The unified data table occupies all of the allocated memory not utilized by other portions of the compiler. If a \$ LIMITS card is not included, approximately 600 data entries can be contained in the available memory area. Each table entry will vary in size from eight to 15 words; most of the variation in size depends upon the length of the data-name. The remainder of the space is used for assigned symbols, alter numbers, address chains, and the properties, usage, size, etc. of the data item.

In the data table, the data entries are chained together via addresses so that either the whole table, or any entity within it, can be traversed. Up to 50 level-number 77 data entries are chained to a dummy level 01 header and a new dummy header is then created, if necessary. As each new data-name is added to the table, the previously stored data entries are traversed. If duplicate data-names are discovered, both are annotated to require qualification if they are referenced in the Procedure Division.

The compiler includes provisions for dynamically expanding the size of the data table. When the data table, as initially allocated, is full, the compiler attempts to obtain additional memory in 4096 (4K) word increments via the MME GEMORE function. If successful, the data table is increased by the amount of memory specified and the source program processing continues. If three denials are received, the data table enters an overflow status and the following discussion is applicable.

If the amount of memory available is not sufficient to contain the entire data table, portions of the table are written as overflow to the \*3 file. The \*3 file is the compiler address table, assigned by the operating system during source program compilation to be used as a temporary working area.

1. When additional memory is required, a level 01 entry, with its subordinate entries, is written from the available memory to the \*3 file.
2. Sufficient level 01 group items are written to the \*3 file so that enough memory is left in the compiler to store the remaining data entries.



To obtain improved compilation performance, the following items should be considered:

1. When each compilation is completed, the compiler prints the number of data table overflow reads and writes. If the numbers are nonzero, consider increasing the allocated memory with a \$ LIMITS card (if recompilations are necessary) until the numbers are zero. The overflow condition increases compilation time because:
  - a. Portions of the data table must be written to a file.
  - b. The portions of the data table that have overflowed must be retrieved if a specified data-name is not in memory.

- c. An 01 group item is read back into the same memory area from which it was written. Thus, the subsequent data entries stored in the same area must also be written out prior to being written over.
  - d. When the overflow mode is used, a data entry must be moved to an alternate memory area since a search for a second operand may result in an overlay of the first operand. Additional table overflow may result if a statement contains numerous operands and the alternate memory area is not sufficient to contain the operands in the statement.
2. If a considerable part of the data table is written to the \*3 file, this file may become full, causing a C1 abort. In this case, either additional links must be allocated to the \*3 file and/or the G\* and \*1 files must be assigned to tape.

#### DATA-NAME/FILLER ITEMS

FILLER is a reserved word; abbreviations such as FILL or FLR should not be used. If an abbreviation is used, it is a data-name and, as such, it participates in all data table searches.

Adjoining FILLER entries should be combined into one entry. This consideration is significant in a table of values where the tendency is usually to make a single entry for each occurrence of the value. One entry in this case could cover several entries or possibly the entire table.

A group having numerous FILLER entries and VALUE clauses should not be constructed to reinitialize another group using the MOVE statement; for example, to reset COMP-4 fields to zero. Rather, the initial values should be specified in the description of the original group, a reinitializing item should be specified containing one entry that is the total size of the group, and the reinitializing item should itself be initialized by a move from the original group at the beginning of the execution of the program.

The length of a data-name governs the amount of space it occupies in the data table and the amount of time required to compare it during a search operation. Unreferenced data-names should be kept to a minimum by changing them to FILLER items and, as noted above, adjoining FILLER entries should be combined. FILLER is always a half-word. Data-names are built into the data table as follows:

1. The first two characters of the data-name and the length of the data-name are stored in one half-word.
2. The remaining characters, if any, occupy as many full words as required, up to a maximum of five full words.

When the data table is searched for a data-name, the size of the data-name and the first two characters of the data-name are compared with the current table entry. If inequality occurs, the search proceeds to the next entry. If equality occurs, the remainder of the data-name is compared, character by character, until an unequal condition occurs or the character-string is exhausted. Variations in length or in spelling at the beginning of the data-name will reduce the time required to perform the search process.

## GROUP ITEMS

The memory space for the data table must be sufficient to contain the 01 level entry having the largest number of data items in its hierarchy. In the case of an implied or specified REDEFINES entry at the 01 level, sufficient data table memory must be available to contain all of the redefined entries.

## ELEMENTARY ITEMS

Elementary items in working-storage should not be defined as level 01 entries. They should either be included as part of a group or be defined as level 77 entries. Figurative constants, such as SPACES, should be used in preference to specifying literal values. If literals are required in the Procedure Division, such as in the construction IF A = "ABC", they should not be defined in the Data Division.

### Procedure Division Entries

The user can also affect the efficiency of the compilation in the Procedure Division. To obtain improved compilation performance, the following items should be considered:

1. Avoid using identical data-names in the Data Division.
2. If possible, avoid specifying multiple receiving fields in excess of six.
3. Avoid using complex conditional statements and compound COMPUTE statements that are both difficult for the compiler to decode and for the user to debug. Instead, write several simple statements to produce the same results.
4. Avoid MOVE CORRESPONDING. Write individual MOVE statements instead.
5. If segmentation is used, design the program so that the segment numbers are in ascending sequence when they are submitted to the compiler. (Otherwise, a separate reordering pass is required during compilation.)

### Resource Allocation

The user can affect the efficiency of the compilation by changing the resources allocated to the compilation. If the quantity of resources is not specified by the user, the following quantities are automatically assigned by the operating system:

|                |                        |
|----------------|------------------------|
| Memory         | - 32,767 (32K)         |
| SYSOUT         | - 20,000 printer lines |
| *3 file        | - 15 random links      |
| Processor time | - .15/hour             |

A report containing the actual results of the compilation and the total number of error, warning, and efficiency messages is printed by the compiler and is different for each compilation. An example of the summary report printed at the end of every compilation is given below.

```
***** THE ABOVE LISTING CONTAINS 000 ERROR MESSAGES *****
*** THE ABOVE LISTING CONTAINS 013 WARNING MESSAGES ***
* THE ABOVE LISTING CONTAINS 000 EFFICIENCY MESSAGES *

COMPILATION TIME (MIN): ELAP CLOCK = 000.62 PROC = 000.35

00000 OVERFLOW READS 00000 OVERFLOW WRITES 27971 WORDS MEMORY USED

00032 LINKS USED ON *3 FILE
```

To determine when the arbitrary (default) resource allocations should be modified by the user, the following guidelines are offered. These guidelines are expressed in terms of the number of COBOL source program lines without regard to the type of coded statement or the computer configuration upon which the compilation is to be executed. The actual requirements for a particular program could vary from the suggested guidelines and can be modified by the user following the initial compilation.

1. Memory - Allocate 32,767 (32K) words of memory for the first 600 source lines in the Data Division and an additional 8192 (8K) words for each additional 800 Data Division lines or portion thereof. The allocation requirements can be determined from the compilation summary report and should be increased if any overflow condition occurs. Do not attempt to compile with less than 32,767 or more than 128,000 words of memory.
2. SYSOUT - If compiling with the standard option NLSTOU included on the \$ COBOL card, allocate two printer lines for each source program line. This assignment allows additional lines for headings, error messages, cross-references, and the compilation summary report. If compiling with the LSTOU option, allocate 15 lines for each source program line.
- \* 3. \*3 File - Allocate 20 random links for each 1000 source lines. The total number of links used on the \*3 file can be determined from the compilation summary report. The total number of source lines can be determined by the highest alter number printed on the compilation listing. (There is one alter number for each source line on the compilation listing.)
4. Processor Time - Allocate 15 hundredths of an hour for each 5000 source program lines or portion thereof.

To compile a program having 6000 source statements, of which 1200 are located in the Data Division, the initial deck setup could be arranged as follows:

| 1      | 8      | 16               |
|--------|--------|------------------|
| \$     | COBOL  | (options)        |
| \$     | LIMITS | 30,40K,,         |
| \$     | FILE   | *3,X1R,150R      |
|        |        | (source program) |
| \$     | ENDJOB |                  |
| ***EOF |        |                  |

### File Utilization

The \*3 file is used extensively by the COBOL compiler and contains several types of data. Data stored on this file at various times during program compilation includes:

1. Data table overflow, if any.
2. Cross-references and error messages.
3. Procedure Division statements transformed to an internal analyzer language.
4. The internal analyzer language transformed to an internal generator language.
5. Parts of the generated GMAP coding of the object program.

If an abnormal termination occurs for large COBOL programs and a C1 abort code is issued, this condition indicates that the \*3 file is full and additional random links must be allocated to obtain a successful compilation.

NOTE: No maximum size limitation is currently imposed on the \*3 file.

Some additional steps may be performed to avoid a C1 abort condition:

1. If any overflow writes are listed in the compilation summary report, increase the memory allocation by 8192 (8K) words and recompile the program.
2. Direct the cross-reference/error sort program to employ a peripheral device other than the \*3 file for merge passes, or delete the cross-reference listing:
  - a. To sort on a different peripheral device, specify one of the following control cards in the source program:

\$ FILE S1,X5R,30R

or

\$ NTAPE S1,C,3



- b. To delete the cross-reference listing, specify the NOXREF option on the \$ COBOL card.
3. Direct some of the internal data to magnetic tape files (if assigned by the user, these files must be tape files) by including the following control cards and recompiling the program:

```

$ TAPE G*,X2R
$ TAPE *1,X3R

```

NOTE: This step should be accomplished as a last resort, since the allocation of the \*1 file and the G\* file to tape may significantly increase compilation time.

If the program still does not compile, prepare and submit a software note and a memory dump to the Honeywell site representative.

An example of the resource allocation required for a very large COBOL source program follows:

```

1 8 16

$ COBOL LSTOU
$ LIMITS 30,75K,,89000
$ FILE *3,X1R,200R
$ TAPE G*,X2R
$ TAPE *1,X3R
$ FILE S1,X4R,30R

 (source program)

$ ENDJOB
***EOF

```

### Compilation Aborts

If the compiler is aborting with an abort code other than C1 and the problem cannot be resolved locally, prepare and submit a software note. It is requested that the information from one of the three options listed below be included with the software note:

1. Recompile the program and generate a special test monitor list/dump. This unique COBOL feature creates a dump listing especially formatted for debugging the COBOL compiler and is prepared as follows:
  - a. Retain the control cards used when the compiler aborted but increase the amount of memory allocated by 5120 (5K) words.

- b. Include the following additional control cards immediately following the \$ COBOL card:

| <u>1</u> | <u>8</u> | <u>16</u>  |
|----------|----------|------------|
| \$       | FILE     | AD,X8R,10L |
| \$       | FILE     | AC,X9R,50L |

or

|    |      |        |
|----|------|--------|
| \$ | TAPE | AC,X9R |
|----|------|--------|

and include the following cards immediately after the last source program cards but preceding the \$ EXECUTE card (if specified):

|    |      |      |
|----|------|------|
| \$ | DATA | CR   |
|    | LIST | DUMP |

\*TMEOF



A sample test monitor list/dump deck setup is shown below:

```
1 8 16

$ COBOL (options)
$ FILE AD,X8R,10L
$ TAPE AC,X9R
$ LIMITS 30,37K
$ FILE *3,X1R,100R

 (source program)

$ DATA CR
 LIST DUMP
*TMEOF
$ ENDJOB
***EOF
```

2. Recompile the program and include the DUMP option in the variable field on the \$ COBOL card.
3. Include a COMDK copy of the source program with the software note, along with any necessary libraries. If the COMDK is submitted on magnetic tape, ensure that it is properly identified by including the name of the submitter, the number of the software note, type of data on the file, method of file preparation, tape density, and tracks.

#### TIME-SAVING AND SPACE-SAVING TECHNIQUES

The following techniques are recommended to obtain efficient COBOL object programs. Consideration is given to input-output, report printing, data manipulation, and data description techniques.

Some of the suggestions are designed to conserve memory space, some are meant to save time, and some will do both. Each recommendation is followed by the designators (T) for time saving, (S) for space saving, or (T and S) for time saving and space saving, to indicate the anticipated type of efficiency.

#### Input-Output Techniques

- The APPLY PROCESS AREA phrase can be used to reduce the number of generated instructions if many statements refer to data items within a file. The amount of memory saved may exceed the memory requirements needed for the 'process area'. Also significant is the amount of execution time saved if the statements referring to the contents of a file are heavily exercised. The implied move between the process area and the buffer, supplied by the File and Record Control program, is very efficient in reducing execution time. The APPLY PROCESS AREA phrase is especially recommended for files containing OCCURS data items. (T and S)
- If it is known in advance that two or more files will not be open concurrently, use the SAME AREA phrase to conserve memory space. This option causes the specified files to utilize the same buffer area(s). (S)

- In a non-mass-storage file maintenance program, consider using the SAME RECORD AREA phrase for the 'throughput' or master file. This option causes an implied process area to be applied to both the input and output versions of the file.

If the assignment of a process area would not be efficient for a throughput file, process the records in the input buffer area (where the READ statement leaves them), and then transmit them to output via WRITE...FROM statements. This approach simplifies the insertion of new records by reducing the number of moves required when a process area is used. The amount of processing to be accomplished on a given record should be considered when deciding whether or not to specify a process area for that record. This method is not allowed when mass storage records are being processed; in this case a process area is required and is supplied automatically by the compiler. (T and S)

- Both READ...INTO and WRITE...FROM imply moves within memory. An explicit or implicit process area also implies a move on each READ or WRITE statement. Therefore, avoid the combination of process area with READ...INTO or WRITE...FROM. For example, the least efficiency results from having a process area for both the input and output master files, and then using READ...INTO a working-storage record area and WRITE...FROM that area:
  - a. Because of the process area, the READ implies a move from buffer to process area.
  - b. The INTO option implies a move from process area to working-storage (in this example).
  - c. The FROM option of WRITE implies a move from working-storage to the output file's process area (in this example).
  - d. Because of the process area, the WRITE implies a move from process area to output buffer.

Four MOVES here have done the work of one. (T and S)

### Incremental Report Printing Techniques

A report printing feature allows large, serially produced reports to be transmitted to a printer in increments, before the end of the activity in which the report is produced is reached. This incremental printing feature is attained by means of the backdoor file and is referred to as 'spinoff'.

A \$ USE .CGSPN card must be included in the card deck to indicate that the spinoff feature is to be used.

A \$ DATA P. card must precede any \$ FUTIL card(s) for this feature. If the P. file is present, its contents are examined for the presence of one or more \$ FUTIL cards. The format of these cards is:

| 1  | 8     | 16           |
|----|-------|--------------|
| \$ | DATA  | P.           |
| \$ | FUTIL | fcl,,DUMP/m/ |

where: fcl - Two-character file-code of the file for which spinoff is desired. The fcl field must not begin with the letter S; this designation is reserved for sort or merge file usage.

m - Number of pages within a given file-code at which spinoff to another printer is to be initiated. The number of pages specified must not exceed 999999.

A maximum of ten files may be designated as spinoff files. The user provides the initial allocation for the file for which spinoff is desired using a \$ FILE card. The format of this control card is:

```
1 8 16

$ FILE fcl,LUD,nnL,device type
```

where: fcl - Two-character file-code of the file for which spinoff is desired. The fcl field must not begin with the letter S; this designation is reserved for sort or merge file usage.

nn - Number of links initially allocated to the file. This allocation must specify a LINKed file (L).

The automatic page counting is controlled by the WRITE ADVANCING... TOP OF PAGE statement in the source program. When the first report request is spun off to SYSOUT for a given file, the least significant character of the file-code is incremented by one. For example, if the original file-code specified on the \$ FUTIL card was Z0, the first spinoff code calculated would be Z1.

If the newly calculated file-code is unique, a new file is created for the spinoff procedure. The number of links requested is equal to the number of links allocated for the original file.

Each file in a CLOSE statement is checked to determine if the file is a spinoff file. If the file is a spinoff file, the final report segment is spun off to SYSOUT. If a denial return is received, an immediate abort is executed with COBOL abort code CH. If the SYSOUT backdoor file is not configured (005 status), an immediate abort is executed with COBOL abort code CI. (T)

An example of the control cards used with the spinoff feature follows:

```
1 8 16

$ USE .CGSPN
$ EXECUTE
$ FILE A1,X1S,10L
$ DATA P.,,COPY
$ FUTIL A1,,DUMP/10/
 .
 .
$ ENDCOPY
$ ENDJOB
```

## Data Manipulation Techniques

- Avoid using the CORRESPONDING option when a simple MOVE statement would suffice. MOVE CORRESPONDING results in a series of moves of individual items; a simple MOVE is instead optimized for the group or record as a whole. Never use MOVE CORRESPONDING for such purposes as transmitting a master file record from the input buffer to the output buffer. Use MOVE CORRESPONDING when it will in fact cause selected items to be moved, or when editing or format conversion is needed on the respective items. (T and S)
- Manipulate a group item or record as a whole whenever possible, rather than manipulating its elementary items separately. This rule is especially important for tables of data items; MOVE or clear a table as a whole whenever possible. For example, technique a. (below) is quite efficient, while b. is very inefficient:
  - a. MOVE SPACES TO TABLE.
  - b.            COMPUTE I = 1.  
      LOOP.        MOVE SPACES TO TABLE-ITEM (I).  
                  COMPUTE I = I + 1.  
                  IF I NOT > TABLE-SIZE GO TO LOOP.(T and S)
- If a subscripted item is to be referred to more than once with the same subscript value(s), consider moving it to a temporary working-storage area just once for all processing. (T and S)
- If a data item is to be used in several subscripts without a change in value, either make it a COMPUTATIONAL-1 item or else move it to a temporary area in working-storage (described as COMPUTATIONAL-1) and use the working-storage data item in the subscripts. (T and S)
- For MOVES, conditions, addition, and subtraction, give the items similar PICTUREs and USAGEs whenever possible. (T)
- In the UNTIL option of the PERFORM statement, use the simplest possible condition to terminate the loop. If necessary, achieve such simplicity by preceding the PERFORM with explicit MOVES and COMPUTES. If numeric items are involved in the condition, give them similar PICTUREs and the same COMPUTATIONAL (-1 or -2) usage, not DISPLAY usage. (T)
- Tend to utilize procedural literals rather than constant values in working-storage. The compiler can optimize the format and word orientation of procedural literals, but must resort to dynamic format conversions in the object program if working-storage items are not ideally aligned. (The user can accomplish ideal alignment by using exceptional care in coding data descriptions, but this practice is not recommended for ordinary constant values.) However, duplicate literals do result in extra memory space requirements. (T)
- Use GO TO...DEPENDING for decisions whenever possible. In any application for which GO TO...DEPENDING can be used, more efficient object coding can be generated than using a succession of IF statements. (T and S)

## Data Description Techniques

- Whenever possible, specify COMPUTATIONAL(-n) usage for a numeric item that will be involved in formulas, arithmetic statements, or numeric comparisons. External considerations sometimes dictate DISPLAY usage; in these cases, consider explicitly moving the item to a COMPUTATIONAL(-n) working-storage area just once for all processing. (Such a move would be inappropriate if only one procedural statement refers to the item.) (T)
- COMPUTATIONAL(-n) usages should be employed in the following preferential order:
  - a. Use COMPUTATIONAL-1 if the item is an integer and is not involved arithmetically with COMPUTATIONAL or COMPUTATIONAL-2 items. (T)
  - b. Use COMPUTATIONAL if the item is not an integer or if it interacts with other COMPUTATIONAL items. Be sure to use COMPUTATIONAL if precise fractional results are required. (T)
  - c. Use COMPUTATIONAL-2 if the advantages of binary floating-point calculations are needed. (The main advantage is the capacity for accurate representation of very large or very small numeric values.) Arithmetic coding for COMPUTATIONAL-2 items is highly efficient, but it does not yield the exact decimal precision needed in most commercial applications. (T)
- Specify COMPUTATIONAL-1 for a data item that will be used as a subscript or that will be a DEPENDING item in a GO TO statement or in an OCCURS clause. This rule is important if the item will be mentioned as a 'subscript-name' in PERFORM...VARYING or in any such loop. Again, consider moving the item explicitly to a COMPUTATIONAL-1 area in working-storage if other considerations dictate USAGE DISPLAY. (T)
- USAGE COMPUTATIONAL-1 is also recommended for identifier-2 data items in WRITE...ADVANCING statements to avoid unnecessary conversions. (T)
- For data items in working-storage that do not specifically require close packing across computer words, specify SYNCHRONIZED. This rule is important for repeated items in a table (OCCURS), and especially for any table which must be repeatedly searched. (T and S)
- If a record contains COMPUTATIONAL(-n) and/or SYNCHRONIZED data items, place single-word items and double-word items together whenever possible. Considerable savings in memory space can be obtained; this rule is most applicable for records in a file. (S)
- For COMPUTATIONAL(-n) data items, ensure that the PICTURE clause is single-word precision, rather than double-word precision, wherever single-word precision is sufficient. (S)
- If the end of a record does not coincide with the end of a word, implied FILLER accounts for the unused character positions. Try to specify explicit FILLER for this purpose, rather than allowing implicit FILLER. (The result is specifically oriented to the Series 60/6000 computer word size.) (T)
- Consider assigning an odd word length to all records in a variable-length file, and an even word length to all records in a fixed-length record file. (T)



- It is often necessary to organize peripheral files in a highly efficient space-saving manner, even though it is also desired to save time while processing the data. In this case, describe each record in both the File Section and in the Working-Storage Section. In the File Section, pack the data as closely as possible, without regard to processing efficiency; in the Working-Storage Section, do exactly the opposite. Avoid using PROCESS AREA (if possible) and avoid READ...INTO and WRITE...FROM. Instead, READ each record and, while it is still in the input buffer, determine whether the record is to be involved in detailed processing. If detailed processing is required, employ the MOVE...CORRESPONDING statement to unpack either the entire record or the significant group(s) within it to the working-storage area and refer to the data in that location for all detailed processing. Similarly, use MOVE...CORRESPONDING as appropriate to construct (or reconstruct) the output record. Perform a simple MOVE from input buffer to output buffer if detailed processing is not required. (T and S)
- If reports are generated without using the Report Writer facility, use skeleton lines in working-storage, with constant information initialized via the VALUE clause rather than by MOVE statements in the Procedure Division. (T and S)

#### USE OF COMPRESSED DECKS

A compressed source program deck may be obtained from a COBOL compilation by specifying either the COMDK or CCOMDK option on the \$ COBOL card. The CCOMDK option specifies that text copied from a library is to be included in the compressed deck. For a complete description of the interaction of the compressed deck options (COMDK, CCOMDK) with the COPY options (COPY, LIBCOPY), refer to Section XIV.

Each card of the compressed deck may be labeled in columns 73-80 by including the desired label in columns 73-80 of the first COBOL source card (IDENTIFICATION DIVISION card). Refer to the CALL, IOEDIT and COMDK Format paragraphs in the File and Record Control reference manual for labeling conventions.

\*

When an American National Standard segmented program is being compiled, with Procedure Division sections not in ascending priority-number order, the compiler rearranges the source program if SEGMNT has been specified in the variable field on the \$ COBOL card. If the CCOMDK option is specified on the \$ COBOL card, the new compressed deck will correspond to the rearranged source program as presented on the source listing. If the COMDK option is specified on the \$ COBOL card, the new compressed deck will not correspond to the rearranged source program as presented on the source listing.

## CROSS-REFERENCE FACILITY

In very large COBOL source programs, the compiler sometimes omits all REF TO entries from the listing, due to a limitation on the number of cross-references and error messages. If the number of cross-references and error messages is more than 16,383, the REF TO entries are omitted. An optional external sort process may be used when memory space is insufficient. The external sort process is not called automatically; one or more collation files (\$ NTAPE S1,A,3 or \$ FILE S1,LUD,nnR, etc.) must be included for use by the sort. If one or more collation files are present, the external sort is called regardless of the size of the source program. The external sort option may be used as a site standard for large source programs. However, the internal sort process is more efficient and therefore the external sort option should be used on an exception basis only.

To facilitate compilation of very large COBOL programs, the cross-references may also be omitted from the source listing by using the NOXREF option in the variable field on the \$ COBOL card. Since valuable information may be deleted when this option is used, it should only be employed as an aid in obtaining successful compilation when ordinary attempts fail.

## PACKED DECIMAL EFFICIENCY TECHNIQUES

Since the Series 60/6000 EIS processors do not operate directly on six-bit character-oriented data, any data item whose PICTURE is numeric and whose usage is DISPLAY must be moved implicitly by the COBOL compiler to a packed decimal format in a temporary work area before it can be used in computations. For example, if the statement ADD A TO B is executed where A and B are described as numeric DISPLAY items, the coding required to accomplish the ADD would be:

1. Move A to a packed format temporary area.
2. Move B to a packed format temporary area.
3. Add A to B using the packed format temporary areas and store the result in a packed format temporary area.
4. Move the resultant packed format temporary area to B using the numeric DISPLAY format.

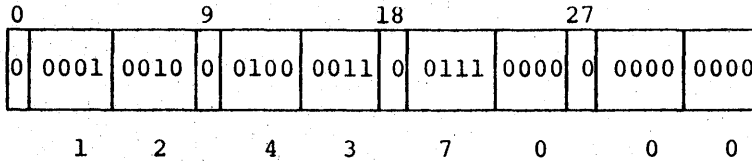
If the statement ADD A TO B is executed, where A and B are described as packed decimal items, the coding required to accomplish the ADD would be:

Add A to B and store the result in B.

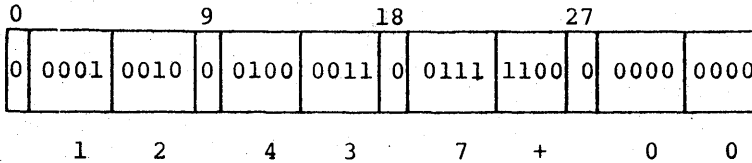
As shown in the above descriptions, object coding efficiency can be attained using packed decimal formats for computations.

The Series 60/6000 COBOL compiler will generate coding to process packed decimal numbers that are either unsigned or contain a separate trailing sign depending on the PICTURE description of the number.

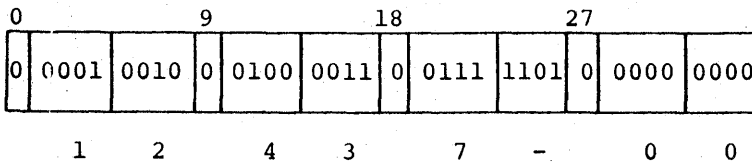
If a packed decimal item is described with a PICTURE 99999 and a VALUE 12437, the field would appear in memory as follows:



If a packed decimal item is described with a PICTURE S99999 and a VALUE 12437, the item would include a separate trailing plus sign (octal 14) in memory as follows:



If a packed decimal item is described with a PICTURE S99999 and a VALUE -12437, the item would include a separate trailing minus sign (octal 15) in memory as follows:



The following suggestions are provided to assist the user in writing accurate and efficient programs using the packed decimal formats:

- Packed decimal cannot be used with a PICTURE clause that contains a 'p'.
- Subscripts must contain a value greater than zero; otherwise, an F0 memory address fault will occur. Erroneous data will be obtained if the subscript value is greater than zero but less than the minimum occurrence number specified in the OCCURS clause.
- Differences in results may occur when using decimal arithmetic as opposed to floating-point arithmetic. The reason for this possible discrepancy is that a floating-point data item occupies an entire Series 60/6000 word, regardless of the PICTURE description, whereas a decimal data item occupies the exact number of character positions indicated in the PICTURE description.

Example:

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 REC-NAME.
02 DN-1 PIC 9V9999 VALUE 12345.
02 DN-2 PIC 9V999 VALUE 6789.
02 DN-3 PIC 99V99.
PROCEDURE DIVISION.
PAR-NAME.
 DIVIDE DN-2 BY DN-1 GIVING DN-3.
 ADD DN-1 DN-2 TO DN-3.
```

Decimal Arithmetic

DN-3 will contain 5.49 after the DIVIDE statement has been executed and 13.51 after the ADD statement has been executed.

Floating-Point Arithmetic

If DN-1, DN-2, and DN-3 are described with USAGE COMP-2, DN-3 will contain 5.49930902... after the DIVIDE statement has been executed and 13.52280902... after the ADD statement has been executed.

- Possible truncation of the results may occur when working with decimal arithmetic rather than floating-point arithmetic. In the previous example, if DN-3 were described with PIC 9V99 rather than PIC 99V99, the following results would be obtained:

Decimal Arithmetic

DN-3 would contain 5.49 after the DIVIDE statement has been executed and 3.51 after the ADD statement has been executed.

Floating-Point Arithmetic

DN-3 would still contain 5.49930902... after the DIVIDE statement has been executed and 13.52280902... after the ADD statement has been executed.

- Group packed decimal fields together in record formats wherever possible to improve memory space utilization.
- Subscripts for arrays should be kept in binary integer format. Packed decimal formats for subscripts are more efficient than numeric DISPLAY, but not as efficient as COMP-1.
- If a packed decimal field is signed, it must be initialized to contain a value with a proper sign. When a numeric descriptor indicates that a field is packed and signed, the sign must be an octal 14 or 15. If the sign is incorrect, an 'ILLEGAL EIS DATA' abort message is printed.
- A field should always be signed (S9(4)) if there is any possibility that the field can become negative. If a sign is not specified, the absolute value is provided in case of a negative condition.
- In an I-D-S program, a packed decimal data description cannot be specified at the 02 level of an I-D-S record description entry; however, the elementary and group items within the 02 field description can be specified with USAGE COMP-4.

SAMPLE BUSINESS PROGRAM

The following is a sample program (UPDATE) illustrating a common type of business application; namely, updating a master file with insertions, deletions, and modifications of file records. The problem definition is oversimplified for the purpose of the example and no claims are made as to its error-free execution within this context. However, liberal use is made of debug statements to follow the processing of input data and they may be helpful as a guide to their use for debugging programs. This program may also be useful as a reference for the more common usages of I-O, PERFORM, MOVE, and IF statements.

```
000000 IDENTIFICATION DIVISION.
000010 PROGRAM-ID. UPDATE.
000020 REMARKS. THIS EXAMPLE UPDATES A MASTER FILE AND
000030 PRODUCES A COMPLETE LISTING OPTIONALLY BY
000040 MEANS OF A CONTROL CARD. CHANGES TO A MASTER
000050 FILE ARE CARD INPUT ALONG WITH CERTAIN
000060 CONTROL CARDS WHICH ALLOW:

000070
000080 1. INSERTIONS AFTER GIVEN SEQUENCE NUMBER
000090 2. MODIFICATIONS OF DATA OR SEQUENCE NUMBERS ONLY
000100 3. DELETES DATA FROM ONE SEQUENCE NUMBER TO ANOTHER AND
000110 INSERTS FOLLOWING DATA CARDS.
000120
000130 CONTROL CARDS ARE ASSUMED TO BE SORTED BY SEQ.# ON INPUT.
000140 SEQUENCE NUMBERS NOT MODIFIED ARE LEFT UNCHANGED.
000150
000160 CONTROL-CARD FORMATS: COL1 7 19
000170 LIS 999999 (LISTING REQUEST CARD WHERE
000180 999999 DEFINES THE SEQ. #
000190 INSERTION INCREMENT)

000200
000210 *IN 999999 (INSERTS FOLLOWING CARDS
000220 AFTER SEQ.# 999999)
000230
000240 *MF NNNNNN TO MMMMMM
000250 (MODIFIES CARD WITH SEQ.# NNNNNN
000260 WITH FOLLOWING CARD-IF PRESENT-
000270 AND CHANGES SEQ.# TO MMMMMM)
000280
000290 *DE NNNNNN THRU MMMMMM

000300
000310 (DELETES CARDS FROM SEQ.# NNNNNN
000320 THRU MMMMMM AND REPLACES WITH ANY
 FOLLOWING CARDS)

000330 ENVIRONMENT DIVISION.
000340 CONFIGURATION SECTION.
000350 SPECIAL-NAMES.
000360 SYSOUT IS TYP, GTIME IS GTME, GLAPS IS GLAP.
000370 COMPILE PHASE1 ONLY WITH ERRORS.
000380 PROCESS ALL DEBUG STATEMENTS.
000390 *NOTE--ABOVE CLAUSE MAY BE VARIED OR REMOVED TO FACILITATE DEBUG
000400 INPUT-OUTPUT SECTION.
000410 FILE-CONTROL.
000420 SELECT CARD-FILE ASSIGN TO CD FOR CARDS.
000430 SELECT LIST-FILE ASSIGN TO LM FOR LISTING.
000440 SELECT OLD-MASTER ASSIGN TO OM.
000450 SELECT NEW-MASTER ASSIGN TO NM.
000460 I-O-CONTROL.
000470 APPLY STANDARD ON OLD-MASTER, NEW-MASTER.

000480 DATA DIVISION.

000490 FILE SECTION.
000500 FD NEW-MASTER LABEL RECORDS STANDARD, DATA RECORD IS MAS-OUT.
```

```

000510 01 MAS-OUT.
000520 02 MLINE PICTURE X(74).
000530 02 MSEQ PICTURE 9(6).
000540 FD OLD-MASTER LABEL RECORDS ARE STANDARD, DATA RECORD IS MAS-IN.

000550 01 MAS-IN.
000560 02 ILINE PICTURE X(74).
000570 02 ISEQ PICTURE 9(6).
000580 FD CARD-FILE LABEL RECORDS STANDARD, DATA RECORD IS CARD-REC.
000590 01 CARD-REC.

000600 02 C-GR.
000610 03 CLINE PICTURE X(74).
000620 03 CSEQ PICTURE 9(6).
000630 02 CR REDEFINES C-GR.
000640 03 CARD-FLD PICTURE X(3) OCCURS 26 TIMES.
000650 03 FILLER PICTURE XX.
000660 02 CS REDEFINES CR.
000670 03 CON-FLD PICTURE 9(6) OCCURS 13 TIMES.
000680 03 FILLER PICTURE XX.
000690 FD LIST-FILE LABEL RECORDS ARE STANDARD, DATA RECORDS ARE
000700 LIST-REC, DATE-REC.
000710 01 LIST-REC.
000720 02 L-FLD PICTURE X(74).
000730 02 LSEQ PICTURE 9(6).
000740 02 FILL PICTURE X(4).

000750 01 DATE-REC.
000760 02 INDENT PICTURE X(28).
000770 02 DATE PICTURE X(8).
000780* * * * *

000790 WORKING-STORAGE SECTION.
000800 77 OLD-MAS-IND PICTURE 9 VALUE 0.
000810 88 OLD-MAS-OPEN VALUE 1.
000820 88 OLD-MAS-CLOSED VALUE 0.
000830 77 LIST-ALL-IND PICTURE 9 VALUE 0.
000840 88 LIST-ON VALUE 1.
000850 88 LIST-OFF VALUE 0.
000860 77 WRITE-IND PICTURE 9 VALUE 0.
000870 88 WRIT-ON VALUE 1.
000880 88 WRIT-OFF VALUE 0.
000890 77 DISPLAY-ERR PICTURE X(6).
000900 77 CURRENT-SEQ PICTURE 9(6) VALUE 0.
000910 77 INCR PICTURE 9(6) VALUE 1.
000920 77 CX PICTURE 9(3) USAGE COMP-1.
000930 77 TIME-2 PICTURE 9(8) USAGE COMP-1.
000940 77 ELAPSED-TIM PICTURE ZZZ9.9999 .
000950 01 CON-REC.
000960 02 CRD-TYP.
000970 03 C-TYP PIC X(12) VALUE " *IN*MF*DE".
000980 02 TYP-ARR REDEFINES CRD-TYP.
000990 03 CON-TYP PICTURE XXX OCCURS 4 TIMES.

001000 01 INDATE.
001010 02 IDATE.
001020 03 MO PICTURE 99.
001030 03 DA PICTURE 99.

001040 03 YR PICTURE 99.
001050 02 TIME-1 PICTURE 9(8) USAGE COMP-1.

001060 01 OTDATE.
001070 02 ODATE.
001080 03 MOT PICTURE 99.
001090 03 FILLER PICTURE X VALUE "-".
001100 03 DAY PICTURE 99.
001110 03 FILLER PICTURE X VALUE "-".
001120 03 YRR PICTURE 99.

001130 PROCEDURE DIVISION.
001140* * * * *

001150 START-UP SECTION. ACCEPT INDATE FROM GTME.
001160 MOVE MO TO MOT. MOVE DA TO DAY. MOVE YR TO YRR.
001170 MOVE TIME-1 TO TIME-2.
001180 OPEN INPUT CARD-FILE, OLD-MASTER,
001190 OUTPUT NEW-MASTER, LIST-FILE.
001200 MOVE 1 TO OLD-MAS-IND.
001210 MOVE SPACES TO INDENT. MOVE ODATE TO DATE.
001220 WRITE DATE-REC BEFORE ADVANCING 2 LINES.
001230 MOVE SPACES TO FILL.
001240 PERFORM READ-CD.
001250 NOTE - THE FOLLOWING IS A LEVEL 1 DEBUG STATEMENT.

```

```

0012601 DIS-1. DISPLAY CARD-REC UPON TYP.
001270 IF CARD-FLD (1) NOT = "LIS" GO TO CONTROL-CARD-TEST
001280 ELSE MOVE 1 TO LIST-ALL-IND.
001290 IF CON-FLD (2) NOT = " " MOVE CON-FLD (2) TO INCR.
001300 READ-CD. READ CARD-FILE AT END GO TO CLOSE-CARD.
001310 NOTE - THE FOLLOWING IS A LEVEL 2 DEBUG STATEMENT.
0013202 DIS-2. DISPLAY CARD-REC UPON TYP.
001330 CONTROL-CARD-TEST. MOVE 2 TO CX.
001340 BR-C. IF CARD-FLD (1) = CON-TYP (CX) GO TO BRANCH.

001350 ADD 1 TO CX IF CX < 5 GO TO BR-C.
001360 MOVE CARD-FLD (1) TO DISPLAY-ERR.
001370 DISPLAY "UNDEFINED CARD ", DISPLAY-ERR, UPON TYP.

001380 GO TO READ-CD.
001390 BRANCH. SUBTRACT 1 FROM CX.
001400 GO TO INS-RTN, MOD-RTN, DEL-RTN DEPENDING ON CX.
001410 STOP "DEAD END ".
001420* *****
001430 INS-RTN SECTION.
001440 II. IF OLD-MAS-OPEN GO TO READ-OLD.
001450 OLD-MAS-ERR. ADD INCR, CON-FLD (2) GIVING CURRENT-SEQ.
001460 MOVE 1 TO WRITE-IND GO TO INSERT-CARD.
001470 READ-OLD. IF WRIT-ON MOVE 0 TO WRITE-IND GO TO TEST-OLD.

001480 READ OLD-MASTER AT END MOVE 0 TO OLD-MAS-IND GO TO II.
001490 TEST-OLD. IF CON-FLD (2) < ISEQ GO TO OLD-MAS-ERR.

001500 PERFORM PASS-AND-WRITE GO TO READ-OLD.
001510 INSERT-CARD. PERFORM READ-CD.
001520 NOTE - THE FOLLOWING IS A LEVEL 2 DEBUG STATEMENT.
0015302 DISPLAY CARD-REC UPON TYP.
001540 IF CARD-FLD (1) = CON-TYP (2) OR CON-TYP (3) OR
001550 CON-TYP (4) GO TO CONTROL-CARD-TEST.
001560 MOVE CURRENT-SEQ TO MSEQ, LSEQ.

001570 M-LIN. MOVE CLINE TO MLINE, L-FLD.
001580 WRIT-2. WRITE MAS-OUT, WRITE LIST-REC AFTER ADVANCING 1 LINE.
001590 END-IN. ADD INCR TO CURRENT-SEQ GO TO INSERT-CARD.
001600* *****
001610 MOD-RTN SECTION.
001620 M1. IF OLD-MAS-OPEN GO TO TEST-READ.
001630 MOD-ERR. MOVE CON-FLD (2) TO DISPLAY-ERR.
001640 DISPLAY "BAD MODIFY CARD FOR ", DISPLAY-ERR UPON TYP.
001650 GO TO READ-CD.
001660 TEST-READ. IF WRIT-ON MOVE 0 TO WRITE-IND GO TO TEST-SEQ.
001670 READ OLD-MASTER AT END MOVE 0 TO OLD-MAS-IND GO TO M1.
001680 TEST-SEQ. IF CON-FLD (2) NOT = ISEQ PERFORM PASS-AND-WRITE

001690 GO TO TEST-READ.

001695 MOVE CON-FLD (2) TO DISPLAY-ERR.
001700 IF CON-FLD (4) NOT = " " MOVE CON-FLD (4) TO
001710 MSEQ, LSEQ GO TO READ-NEXT-CARD
001720 ELSE MOVE ISEQ TO MSEQ, LSEQ.
001730 READ-NEXT-CARD. READ CARD-FILE AT END PERFORM MOD-WARN
001740 GO TO CLOSE-CARD.
001750 NOTE - THE FOLLOWING IS A LEVEL 3 DEBUG STATEMENT.
0017603 DISPLAY CARD-REC UPON TYP.
001770 IF CARD-FLD (1) = CON-TYP (2) OR CON-TYP (3) OR
001780 CON-TYP (4) PERFORM MOD-WARN
001790 GO TO CONTROL-CARD-TEST.
001800 IF CSEQ NOT = " " MOVE CARD-REC TO MAS-OUT, LIST-REC

001805 PERFORM WRIT-2
001810 ELSE PERFORM M-LIN THRU WRIT-2.
001820 GO TO READ-CD.
001830* *****

001840 DEL-RTN SECTION.
001850 D1. IF OLD-MAS-OPEN GO TO READ-TEST.

001860 DEL-ERR. MOVE CON-FLD (2) TO DISPLAY-ERR.

001870 DISPLAY "BAD DELETE CARD FOR ", DISPLAY-ERR, UPON TYP.
001880 MOVE 1 TO WRITE-IND GO TO READ-CD.
001890 READ-TEST. IF WRIT-ON MOVE 0 TO WRITE-IND GO TO SEQ-TEST.
001900 READ OLD-MASTER AT END MOVE 0 TO OLD-MAS-IND GO TO D1.

```

```

001910 SEQ-TEST. IF CON-FLD (2) > ISEQ PERFORM PASS-AND-WRITE
001920 GO TO READ-TEST.

001930 IF CON-FLD (2) < ISEQ GO TO DEL-ERR.

001940 PASS-NO-WRITE. READ OLD-MASTER AT END MOVE 0 TO OLD-MAS-IND
001950 GO TO OLD-MAS-ERR.
001960 IF CON-FLD (4) > ISEQ OR = ISEQ GO TO PASS-NO-WRITE

001970 ELSE MOVE 1 TO WRITE-IND MOVE CON-FLD (2) TO CURRENT-SEQ
001980 GO TO INSERT-CARD.
001990 *****
002000 PASS-AND-WRITE SECTION. MOVE MAS-IN TO MAS-OUT, LIST-REC.
002010 WRITE MAS-OUT IF LIST-OFF GO TO P1.
002020 WRITE LIST-REC AFTER ADVANCING 1 LINE.
002030 P1. MOVE ISEQ TO CURRENT-SEQ.
002040 NOTE - THIS IS EXIT.
002050 *****
002060 MOD-WARN SECTION.
002070 DISPLAY "NO MODIFY CARD TEXT FOUND FOR" DISPLAY-ERR,
002080 UPON TYP. MOVE ILINE TO MLINE, L-FLD.
002090 PERFORM WRIT-2.
002100 NOTE - THIS IS EXIT.
002110 *****
002120 CLOSE-CARD SECTION. CLOSE CARD-FILE IF OLD-MAS-CLOSED GO TO DONE.
002130 COPY-OLD. IF WRIT-ON MOVE 0 TO WRITE-IND GO TO C2.

002140 READ OLD-MASTER AT END MOVE 0 TO OLD-MAS-IND GO TO DONE.

002150 C2. PERFORM PASS-AND-WRITE GO TO COPY-OLD.
002160 DONE. CLOSE OLD-MASTER, NEW-MASTER, LIST-FILE.
002170 ACCEPT INDATE FROM GTIME.

002180 COMPUTE ELAPSED-TIM ROUNDED = (TIME-1 - TIME-2) / 64000.
002190 DISPLAY "ELAP.CLOCK TIME(SEC)= ", ELAPSED-TIM, UPON TYP.
002200 ACCEPT TIME-2 FROM GLAP.
002210 DIVIDE 64000 INTO TIME-2 GIVING ELAPSED-TIM.
002220 DISPLAY "ELAP.PROC.TIME(SEC) = ", ELAPSED-TIM, UPON TYP.
002230 STOP RUN.
002240 END OF PROGRAM.

```





## SECTION XVII

### OBSOLETE LANGUAGE ELEMENTS

Some of the elements described in this section have been deleted from the COBOL language specifications but are still supported by the COBOL compiler in their last state of implementation. Other language elements contained in this section represent extra-standard features offered in previous versions of the COBOL compiler; they are also supported in their last state of implementation. The use of any of these elements in new programs is strongly discouraged since the transferability of a program from one computer to another may thus be adversely affected. A listing of the obsolete language features follows:

6000 WITH EIS phrase

PREPARED option

Constant Section

CLASS clause

Editing clauses

FILE CONTAINS clause

POINT LOCATION clause

RANGE clause

SEQUENCED clause

SIGNED clause

SIZE clause

USAGE COMP-3 PACKED SYNC clause

Conditional statement IF...OTHERWISE...THEN...

The detailed syntax for the obsolete language elements follows.



ENVIRONMENT DIVISION ELEMENTS

6000 WITH EIS Phrase

The 6000 WITH EIS phrase was formerly used in Format 2 of both the SOURCE-COMPUTER paragraph and the OBJECT-COMPUTER paragraph to designate the computer upon which the program is to be compiled.

General Format:

[ 6000 [WITH EIS] ]

Rules:

1. When the 6000 WITH EIS entry is used in the SOURCE-COMPUTER paragraph or in the OBJECT-COMPUTER paragraph, the computer-name specified should be 6000 or 6000 WITH EIS. Series 60 users should specify 6000 WITH EIS.

DATA DIVISION ELEMENTS

PREPARED Option

The PREPARED FOR computer-name option is specified in the Data Division only when the data descriptions have been written for a computer other than the object computer.

General Format:

DATA DIVISION. [ PREPARED FOR computer-name ]

This option is treated as documentation only.

## CONSTANT SECTION

The Constant Section is organized in the same manner as the Working-Storage Section, beginning with a section header, followed by data description entries for noncontiguous constants, and then followed by data description entries, in that order. A skeletal format for the Constant Section follows:

General Format:

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

The concepts of literals and figurative constants enable the user to specify the value of a constant by writing its actual value (or a figurative representation of that value). In coding, it is often desirable to name this value and then refer to it by name.

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

Constant storage is the memory area that is reserved to save named constants for use in a given program.

### Noncontiguous Constant Storage

Constants that have no relationship to one another need not be grouped into records provided they do not need to be further subdivided. Such items are called noncontiguous constants. Each of these constants is defined in a separate data description entry which begins with a special level-number 77.

The following data description clauses are required in each level 77 entry:

1. level-number.
2. data-name.
3. PICTURE.
4. VALUE.

The OCCURS and REDEFINES clauses are not meaningful for level 77 entries and will cause an error at compilation time if used. Other data description clauses are optional and can be used to complete the description of the constant when necessary.

### Constant Records

Named constants in the Constant Section that have a definite relationship to one another must be grouped into records according to the rules for formation of record descriptions. All record description clauses can be used in a Constant Section record description, including REDEFINES, OCCURS, and COPY. Each Constant Section record-name (01 level) must be unique since it cannot be qualified by a file-name or section-name. Subordinate data-names need not be unique if they can be made unique by qualification.

### VALUE of Constants

In the definition of constants, the VALUE clause is required. VALUE cannot be specified in a group entry that contains items having different usages. All the rules for the expression of literals and figurative constants apply. The size of a literal used to specify the value of a constant can be equal to or less than the specified size of the item, but it cannot be greater.

### Condition-Names

Since a constant can have only one value, there can be no associated condition-names. The use of a condition-name entry (level 88) in the Constant Section is, therefore, illegal and will constitute an error in the source program.

### Tables of Constants

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

1. The table may be described as a record by a set of contiguous record description entries, each of which specifies the value of an element, or part of an element, of the table. In defining the record and its elements, any record description clause (i.e., SIZE, USAGE, PICTURE, editing information, etc.) may be used to complete the definition where required. This form is required when the elements of the table require separate handling due to synchronization, usage, etc. The structure of the table is then shown by use of the REDEFINES entry and its associated subordinate entries.
2. When the elements of the table do not require separate handling, the value of the entire table may be given in the entry that defines the entire table. The lower level entries will show the hierarchical structure of the table.

## DATA DESCRIPTION ENTRIES

### CLASS Clause

The CLASS clause is used to indicate the type of data being described.

#### General Format:

CLASS IS {  
    ALPHABETIC  
    NUMERIC  
    ALPHANUMERIC  
    AN  
}

#### Rules:

1. AN is an acceptable abbreviation for ALPHANUMERIC.
2. The CLASS clause can be written at any level. If the CLASS clause is written at a group level, it applies to each elementary item in the group. The CLASS of an item cannot contradict the CLASS of a group to which the item belongs. ALPHABETIC or NUMERIC items within an ALPHANUMERIC group are not considered contradictory.
3. NUMERIC describes data composed of the numerals 0-9, with or without an operational sign. If there is no sign associated with a NUMERIC item, the item is considered positive. If the item is NUMERIC and no assumed decimal point is indicated, the item is considered positive. If the item is NUMERIC and no assumed decimal point is indicated, the item is considered to be an integer.
4. ALPHABETIC describes data that contains any combination of the twenty-six (26) letters of the Roman alphabet and the space. No other characters can be used.
5. ALPHANUMERIC describes data that may contain any allowable character in the object computer's character set, including alphabets and numerics. Thus, data which is ALPHABETIC or NUMERIC is also ALPHANUMERIC.
6. If both the PICTURE clause and the CLASS clause are given in the same entry, the class of characters shown in PICTURE must not contradict the CLASS clause of an elementary item, or of a group to which the item belongs.
7. If the CLASS of an elementary item cannot be determined from any clause in the item's data description or from the data description of any group to which the item belongs, the CLASS of the item is assumed to be ALPHANUMERIC.

8. An item's CLASS may be implied by various other clauses. If a combination of such clauses appears, either within an entry or between an entry and a group that contains it, they must not contradict each other.

a. The CLASS is NUMERIC if any of the following clauses appear:

CLASS NUMERIC  
 USAGE COMPUTATIONAL  
 USAGE COMPUTATIONAL-n  
 SIZE integer NUMERIC  
 SIZE integer COMPUTATIONAL  
 SIZE integer COMPUTATIONAL-n  
 SIGNED  
 PICTURE containing no characters other than 0s, 9s, Ps, S, and V.

b. The CLASS is ALPHABETIC if any of the following clauses appear:

CLASS ALPHABETIC  
 SIZE integer ALPHABETIC  
 PICTURE containing only As or a combination of As and Bs.

c. The CLASS is ALPHANUMERIC if any of the following clauses appear:

CLASS ALPHANUMERIC (or AN)  
 SIZE integer ALPHANUMERIC (or AN)  
 PICTURE containing X  
 PICTURE containing 9 as well as either A or B  
 PICTURE containing both A and zero (0).  
 PICTURE containing A, \*, \$, comma (,), decimal point (.), -, +, CR, or DB.

### Editing Clauses

The editing clauses are used to permit suppression of nonsignificant zeros and commas, and to permit floating dollar signs or check protection.

General Format:

$$\left\{ \begin{array}{ll} \underline{\text{ZERO}} & \underline{\text{SUPPRESS}} \\ \underline{\text{CHECK}} & \underline{\text{PROTECT}} \\ \underline{\text{FLOAT}} & \underline{\text{DOLLAR}} & \underline{\text{SIGN}} \end{array} \right\} \left[ \underline{\text{LEAVING}} \text{ integer } \left\{ \begin{array}{l} \text{PLACES} \\ \text{PLACE} \end{array} \right\} \right]$$

Rules:

1. The editing clauses can be specified only at the elementary item level.
2. The rules for editing specify that data items are moved in conformity with the record description of the receiving item.



3. The three options, ZERO SUPPRESS, CHECK PROTECT, and FLOAT DOLLAR SIGN, all permit suppression of leading zeros and commas. If the LEAVING option is not employed, suppression will stop as soon as either a nonzero digit or the decimal point (actual or assumed) is encountered. Specifically:
  - a. When ZERO SUPPRESS is specified, leading zeros and commas will be replaced by spaces.
  - b. When CHECK PROTECT is specified, leading zeros and commas will be replaced by asterisks.
  - c. When FLOAT DOLLAR SIGN is specified, the rightmost character suppressed will be replaced by a dollar sign, and all other characters which are suppressed will be replaced by spaces.
4. The LEAVING phrase may be employed to stop suppression before the decimal point (actual or assumed) is encountered. 'Integer' must be a numeric literal with an integral value. When used, suppression stops (leaving integer positions to the left of the real or assumed decimal point) unless stopped sooner by the conditions specified in rule 3. The integer position is a count of the number of characters, starting immediately at the left of the actual or assumed decimal point.
5. When any of these clauses are used, the item is assumed to be alphanumeric because the \$, \*, and + are alphanumeric.
6. More comprehensive editing features are available in the PICTURE clause. When any of the above options are used, the format of the item is assumed to contain editing symbols.
7. If both a PICTURE and editing clauses are present for an entry, the PICTURE clause takes precedence.

#### FILE CONTAINS Clause

The FILE CONTAINS clause is used to document the approximate number of logical records in a file.

#### General Format:

FILE CONTAINS ABOUT integer-1 RECORDS

#### Rules:

1. The FILE CONTAINS clause has no effect on the object program.

## POINT LOCATION Clause

The POINT LOCATION clause is used to define an assumed decimal point or binary point.

General Format:

$$\underline{\text{POINT LOCATION}} \text{ IS } \left\{ \begin{array}{c} \underline{\text{LEFT}} \\ \underline{\text{RIGHT}} \end{array} \right\} \text{ integer } \left\{ \begin{array}{c} \underline{\text{PLACES}} \\ \underline{\text{BITS}} \end{array} \right\}$$

Rules:

1. This optional clause may be used only at the elementary item level.
2. The decimal point location is normally defined with the PICTURE clause, but an assumed decimal point may be specified via the PLACES option of the POINT clause. If both are specified, they must agree.
3. The POINT clause indicates the position of an assumed decimal point only, never the position of an actual decimal point. Actual decimal points may only be specified with a PICTURE clause.
4. When the PLACES option is used, the assumed decimal point is 'integer' decimal places to the left or right of the least significant position of the item. PLACES is implied when neither PLACES nor BITS is specified.
5. When the BITS option is specified, it is used for documentation only.
6. If USAGE is COMPUTATIONAL-n, the point location must not be RIGHT and 'integer' must not exceed the size of the item.

## RANGE Clause

The RANGE clause is used in a data description entry to indicate the potential range of the value of an item.

General Format:

$$\underline{\text{RANGE}} \text{ IS literal-1 } \left\{ \begin{array}{c} \underline{\text{THRU}} \\ \underline{\text{THROUGH}} \end{array} \right\} \text{ literal-2}$$

Rules:

1. The RANGE clause may be specified only at the elementary item level.
2. Literal-1 and literal-2 may be figurative constants.

3. The literals must not contain more digits than are specified in the PICTURE clause.
4. The words THRU and THROUGH are equivalent.
5. The RANGE clause is used only to document the source program.
6. For numeric items, literal-1 and literal-2 represent the respective minimum and maximum values of the item.
7. For nonnumeric items, each character of literal-1 and of literal-2 represents the respective minimum and maximum values of the corresponding character position in the item.

### SEQUENCED Clause

The SEQUENCED clause is used in an FD entry to indicate the keys on which data records are sequenced.

General Format:

SEQUENCED ON data-name-1 [ , data-name-2 ] ...

Rules:

1. The SEQUENCED clause is used only to document the source program and has no effect on object program activities; it cannot be used to cause an automatic sequence check.
2. Data-name-1 represents the major key, data-name-2 represents the next highest key, etc.
3. The data-names should be qualified when necessary, but subscripting is not permitted.

### SIGNED Clause

The SIGNED clause is used to specify the presence of a standard operational sign in an elementary item.

General Format:

SIGNED

Rules:

1. The SIGNED clause may be specified only at the elementary item level.
2. An item whose data description specifies an operational sign must be numeric. Therefore, its CLASS need not be specified.
3. SIGNED indicates the presence of a standard operational sign. (The sign may also be indicated by using an 'S' in the PICTURE clause.) A standard operational sign is not considered in determining the size of the item.
4. An item that contains any editing symbols other than 0 (zero) cannot have an operational sign.
5. The operational sign is contained in the least significant character (two high-order bits) of a signed numeric data item.

SIZE Clause

The SIZE clause is used to specify the size of an item in terms of the number of standard data format characters.

General Format:

SIZE IS integer  $\left\{ \begin{array}{l} \text{CHARACTER [ S ]} \\ \text{DIGIT [ S ]} \end{array} \right\}$

Rules:

1. The size of the data item must be specified at the elementary item level by means of a SIZE clause or a PICTURE clause. A PICTURE clause and a SIZE clause need not both be given, but if both are specified they must agree.
2. If the SIZE clause is specified at any level other than the elementary item level, it is optional.
3. If the SIZE clause is specified at a group level, the size of the group is the sum of the sizes (as determined from SIZE or PICTURE) of the elementary items of which the group is composed.
4. 'Integer' represents the exact number of characters, excluding operational signs.
5. Any of the keywords of the USAGE and/or CLASS clauses can be inserted between integer and the word CHARACTERS (or DIGITS) in the SIZE clause format. If this method is followed, separate USAGE and/or CLASS clauses must not be written.

Example:

02 PAGE-NO; SIZE IS 7 NUMERIC COMPUTATIONAL DIGITS; VALUE IS 0000342.

## USAGE COMP-3 PACKED SYNC Clause

The COMP-3 PACKED SYNC phrase was formerly included as Format 2 of the USAGE clause to indicate packed decimal data items. Since this syntax has now been replaced by the USAGE COMPUTATIONAL-4 format, it is considered obsolete.

### General Format:

$$\left[ \begin{array}{c} \text{USAGE} \\ \text{IS} \end{array} \right] \left\{ \begin{array}{c} \text{COMPUTATIONAL-3} \\ \text{COMP-3} \end{array} \right\} \left\{ \begin{array}{c} \text{PACKED} \\ \text{PK} \end{array} \right\} \left\{ \begin{array}{c} \text{SYNCHRONIZED} \\ \text{SYNC} \end{array} \right\}$$

### Rules:

1. The 6000-EIS or 6000 WITH EIS phrase must be specified in the OBJECT-COMPUTER paragraph.
2. COMPUTATIONAL-3 PACKED SYNCHRONIZED signifies that two four-bit digits will occupy one nine-bit byte (packed decimal). Data in this format can be processed only on a Series 60/6000 Extended Instruction Set (EIS) processor. If the PICTURE character-string specifies an operational sign, the item will be one four-bit digit larger than the number of '9's in the PICTURE character-string would imply.
3. In order to obtain a packed decimal data item, both of the words PACKED (or PK) and SYNCHRONIZED (or SYNC) must be specified, in that order, following COMPUTATIONAL-3 (or COMP-3). The item will be treated as a normal COMPUTATIONAL-3 (or COMP-3) item if neither word is specified.
4. If the USAGE COMPUTATIONAL-3 PACKED SYNCHRONIZED clause is stated at a group level, each elementary item within the group will be treated as packed decimal. The group item itself will be considered to be alphanumeric.

Conditional Statements

In the Procedure Division, the conditional sentence format using the OTHERWISE phrase has been deleted. The syntax of this feature is given below:

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

In the conditional sentence above, the condition is an expression which is true or false. If the condition is true, then statement-1 is executed and control is transferred to the next sentence. If the condition is false, statement-2 is executed and then control is passed to the next sentence.

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

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

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

Statement-3 can in turn be either imperative or conditional and, if conditional, can in turn contain conditional statements in arbitrary depth. In an identical manner, statement-4 can be either imperative or conditional, as can statement-2.

The execution of the OTHERWISE NEXT SENTENCE phrase causes a transfer of control to the next sentence as written, except when it appears in the last sentence of a procedure being performed, in which case control is passed to the return mechanism.

THEN Separator

Prior versions of COBOL allowed the separator THEN to be used between statements in the same manner that the semicolon is currently used. This syntax construction was normally specified in a conditional statement between the condition and statement-1 or between statement-1 and the word OTHERWISE. For example, the conditional sentence could have the form:

IF condition THEN imperative-statement-1 THEN OTHERWISE  
imperative-statement-2.



APPENDIX A

ORDER OF COBOL SOURCE PROGRAM

An outline of the four divisions of a COBOL source program is given below. The grouping within each division is indicated by indention as follows:

| <u>Division</u>                | <u>Required (X)</u> |
|--------------------------------|---------------------|
| <u>Section</u>                 |                     |
| <u>Paragraph</u>               |                     |
| IDENTIFICATION DIVISION.       | X                   |
| PROGRAM-ID.                    | X                   |
| AUTHOR.                        |                     |
| INSTALLATION.                  |                     |
| DATE-WRITTEN.                  |                     |
| DATE-COMPILED.                 |                     |
| SECURITY.                      |                     |
| REMARKS.                       |                     |
|                                | } any order         |
| ENVIRONMENT DIVISION.          | X                   |
| CONFIGURATION SECTION.         |                     |
| SOURCE-COMPUTER.               |                     |
| OBJECT-COMPUTER.               |                     |
| SPECIAL-NAMES.                 |                     |
| INPUT-OUTPUT SECTION.          |                     |
| FILE-CONTROL.                  |                     |
| I-O-CONTROL.                   |                     |
| DATA DIVISION.                 | X                   |
| FILE SECTION.                  |                     |
| input files.                   |                     |
| output files.                  |                     |
| sort files.                    |                     |
| merge files.                   |                     |
|                                | } any order         |
| WORKING-STORAGE SECTION.       |                     |
| REPORT SECTION.                |                     |
| PROCEDURE DIVISION.            | X                   |
| DECLARATIVES. <sup>1</sup>     |                     |
| (USE Sections)                 |                     |
| END DECLARATIVES. <sup>1</sup> |                     |
| (all other procedures)         |                     |
| END PROGRAM.                   |                     |

---

<sup>1</sup>If used, both DECLARATIVES and END DECLARATIVES must be present.





## APPENDIX B

### COBOL DECK SETUPS

Appendix B contains the deck setups for compiling, compiling and executing, executing, creating a tape library using the Bulk Media Conversion (BMC) routines, and compiling using the library facility.

For complete details on all control cards, refer to the Control Cards reference manual.

An example of the \$ COBOL card with options (operand field) is shown below:

```
1 8 16

$ COBOL options
```

The \$ COBOL card is used to bring the COBOL compiler into operation.

By the use of options on the \$ COBOL control card, the user can modify the source program processing. For example, options can be requested that limit or extend the amount of processing to be accomplished by the compiler, that request additional output information, or that specify the form of compiler input and output.

The options are specified in the operand field on the \$ COBOL card. The following options are available (default options are underlined):

- NDUMP - Only program registers will be dumped if the compilation activity terminates abnormally.
- DUMP - A slave memory dump will be produced if the compilation activity terminates abnormally.
- LSTIN - A listing of source input will be produced by the COBOL compiler.
- NLSTIN - No listing of source input will be produced; however, a listing of fatal error messages (\*\*\*\*\*), if any, will be produced.
- NLSTOU - No listing of the assembled object program output will be prepared.
- LSTOU - A listing of the assembled object program output will be prepared by GMAP.

- JREST - Enable job restart.
- NJREST - Do not restart this job.
- REST - Enable activity restart.
- NREST - Do not restart this job with the current activity.
  
- DECK - A binary object program deck will be prepared as output by GMAP.
- NDECK - No binary object program deck will be prepared.
  
- NCOMDK - No compressed deck version of the source program will be prepared.
- COMDK - A compressed deck version of the source program will be prepared, excluding any source lines copied from a COBOL library.
- CCOMDK - A compressed deck version of the source program as presented on the listing, including any source lines copied from a COBOL library, will be prepared.
  
- NCOPY - No copy prepass is performed.
- COPY - A copy prepass is performed, allowing the user to copy source lines from an external library, utilize internal program copies, and to specify the RENAMING phrase in the FILE-CONTROL paragraph. The presence of a library file is assumed during compilation.
- LIBCPY - This option is required when any American National Standard COPY clauses or statements are used in a source program. The presence of a library file is assumed during compilation.
  
- NEISF - Directs the compiler to specifically exclude the use of the Extended Instruction Set (EIS) in the generated object program, overriding the 6000-EIS phrase in the OBJECT-COMPUTER paragraph of the source program.
- EISF - Directs the compiler to specifically include the use of the EIS in the generated object program. This option is equivalent to specifying the 6000-EIS phrase in the OBJECT-COMPUTER paragraph of the source program.
  
- COMP1 - A source listing with cross-references and error messages (if any) is produced at the end of phase 1 of the compilation and the compilation process stops. No object program or assembly listing is produced.
  
- FLAGNA - Source program lines containing features not included in American National Standard COBOL (X3.23-1968) are flagged on the output listing with the prefix 'NA' preceding the reference number.
  
- NOXREF - No source listing cross-references are produced. This option may be included to avoid C1 aborts when compiling large programs.
  
- ON6 - A REF ON control pseudo-operation is generated by the COBOL compiler and a Symbol Reference Table is thereby appended to the GMAP listing.

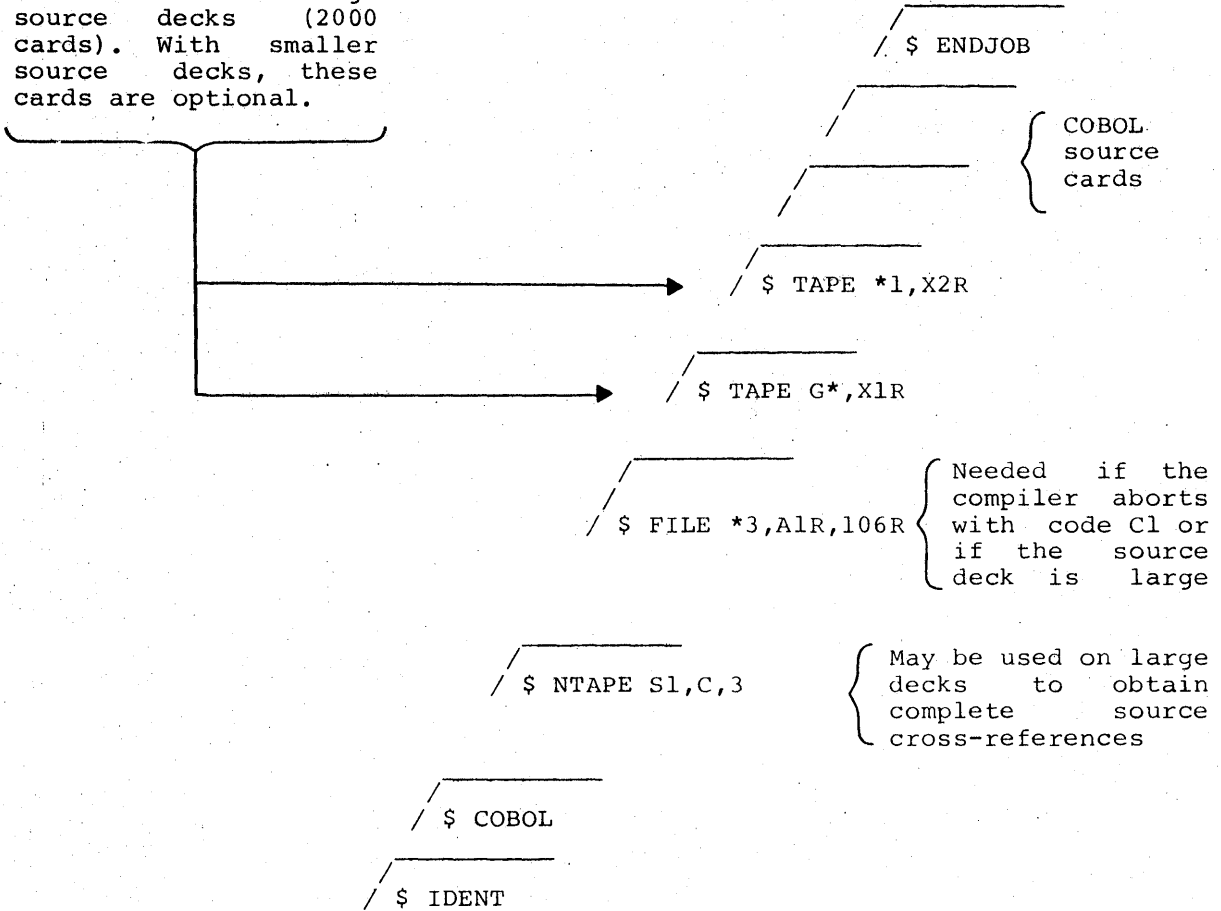
- SYMTAB - GMAP will prepare a listing of the Symbol Reference Table (if one has been built), even though the NLSTOU option has also been specified.
- SEGMNT - This option causes the compiler to reorder the source program Procedure Division sections into ascending priority-number sequence.
- CBL68 - Invokes the compilation of those language elements represented in the highest level of implementation of the COBOL language as specified in American National Standard COBOL X3.23-1968. This option also allows the compilation of certain language elements based on later editions of the CODASYL COBOL Journal of Development or in American National Standard COBOL X3.23-1974. When the CBL68 mode of the compiler is used, the appropriate manuals to be referenced are the Standard COBOL-68 Reference Manual, DE17, and the Standard COBOL-68 User's Guide, DE18.
- SUBSET - Invokes the compilation of a restricted subset of COBOL language elements. In general, this subset is based on pre-1968 editions of the CODASYL COBOL Journal of Development. In the SUBSET mode, some language elements may not function in the same manner as in the CBL68 mode.

Rules:

1. The \$ COBOL card must precede the source program cards of every COBOL program or subprogram to be processed and must precede any other control card associated with that compilation activity.
  2. The options can be listed in any order in the operand field, starting in column 16.
  3. If an option is not specified in the operand field, the corresponding underlined default option, if any, is assumed.
  4. Options may be continued by using the \$ ETC card.
  5. Standard activity limits are predefined. However, if these limits are to be modified, a \$ LIMITS card must immediately follow the \$ COBOL card (unless the \$ ETC card is specified, in which case the \$ LIMITS card must immediately follow the \$ ETC card).
  6. If the NREST option is encountered in the first activity of a job, processing proceeds as if NJREST had been encountered.
  7. A blank terminates the scan of the operand field.
  8. If a source deck is to utilize the HIS COPY function, the COPY option must be specified on the \$ COBOL card. If a source deck is to utilize the American National Standard COPY function, the LIBCPY option must be specified on the \$ COBOL card. See Section XIV.
- NOTE: If both of the above options are specified, the LIBCPY option overrides the COPY option.
9. The CCOMDK option will override the COMDK option if the COMDK option is also specified, so that only one compressed source deck is obtainable from a given compilation.

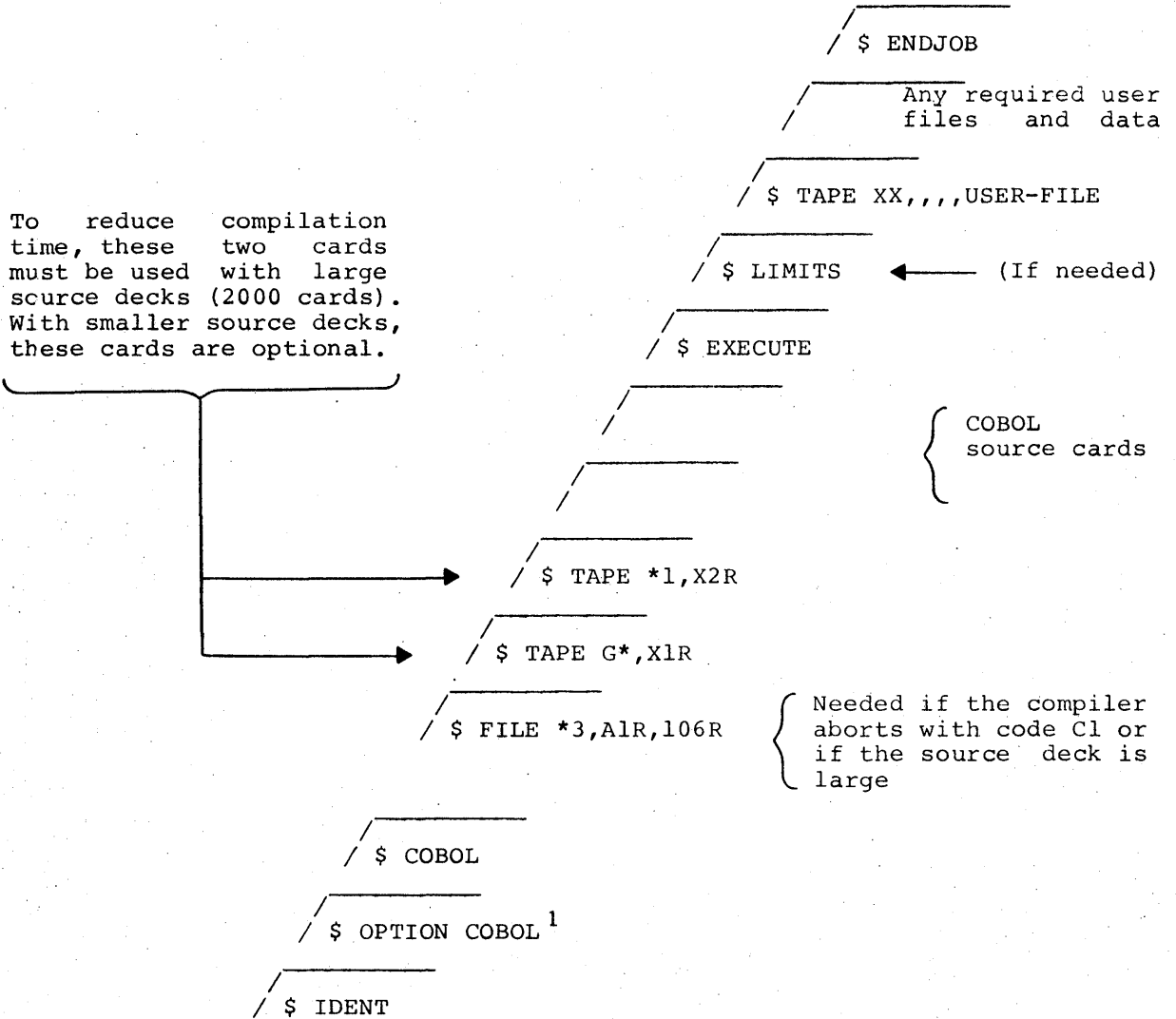
COMPILE ONLY

To reduce compilation time, these two cards must be used with large source decks (2000 cards). With smaller source decks, these cards are optional.



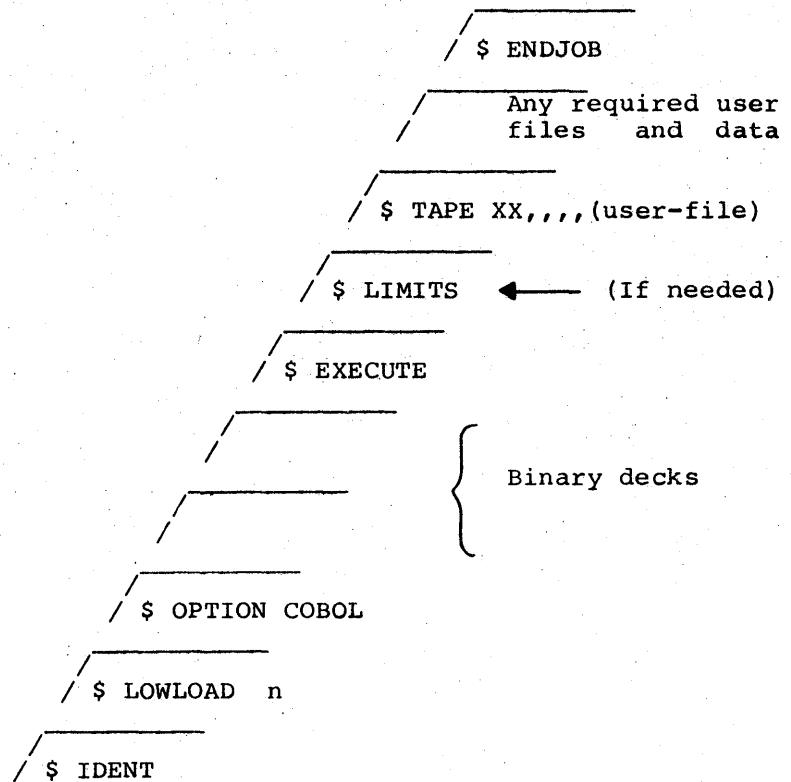
An I-D-S/COBOL translate and compile deck is set up similarly and the \$ COBOL card is replaced with the \$ IDS card.

COMPILE AND EXECUTE



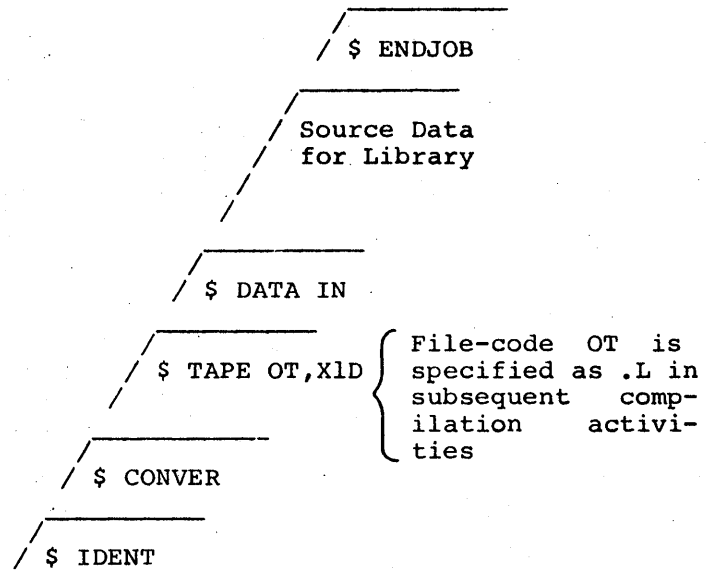
<sup>1</sup>A \$ OPTION COBOL card should be specified at object program execution to ensure that unnecessary subroutines are not loaded.

EXECUTE ONLY

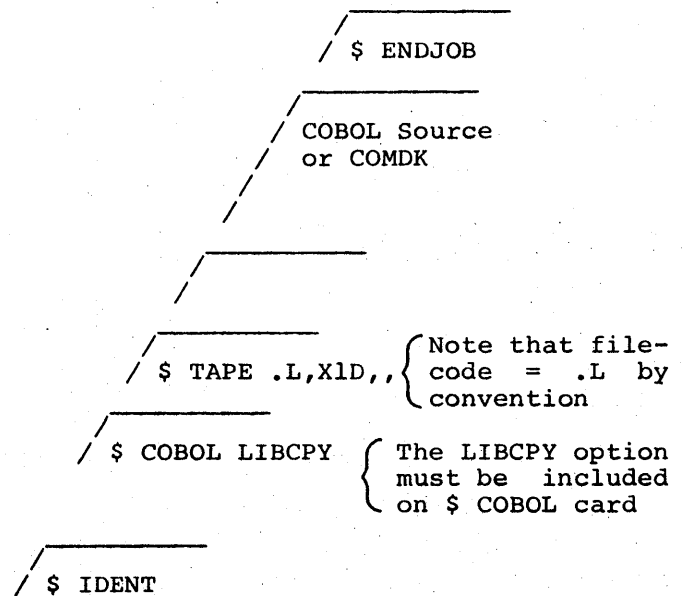


If a \$ LOWLOAD card is required for sharing Blank Common with the loader, the \$ LOWLOAD card must precede the \$ OPTION COBOL card.

CREATING A TAPE LIBRARY (USING BMC)

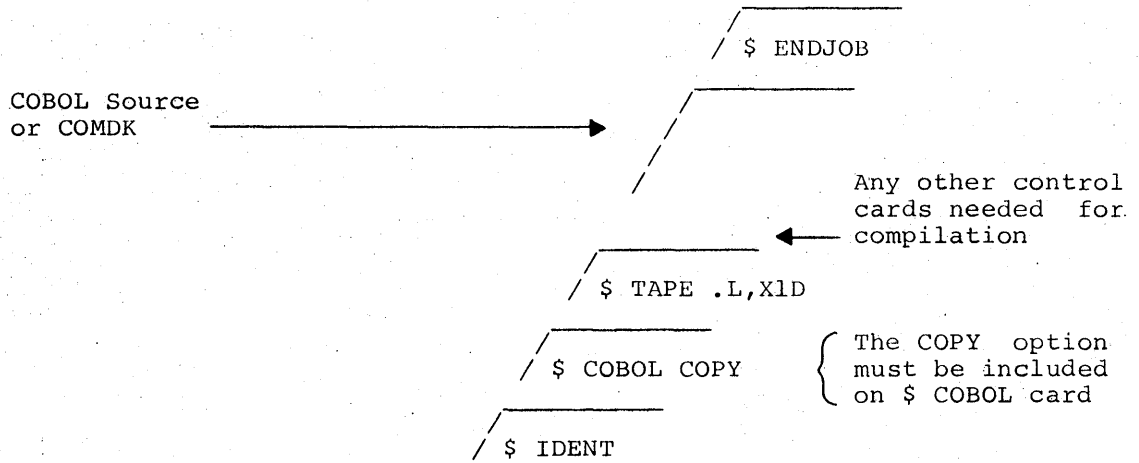


COMPILE USING AMERICAN NATIONAL STANDARD COPY

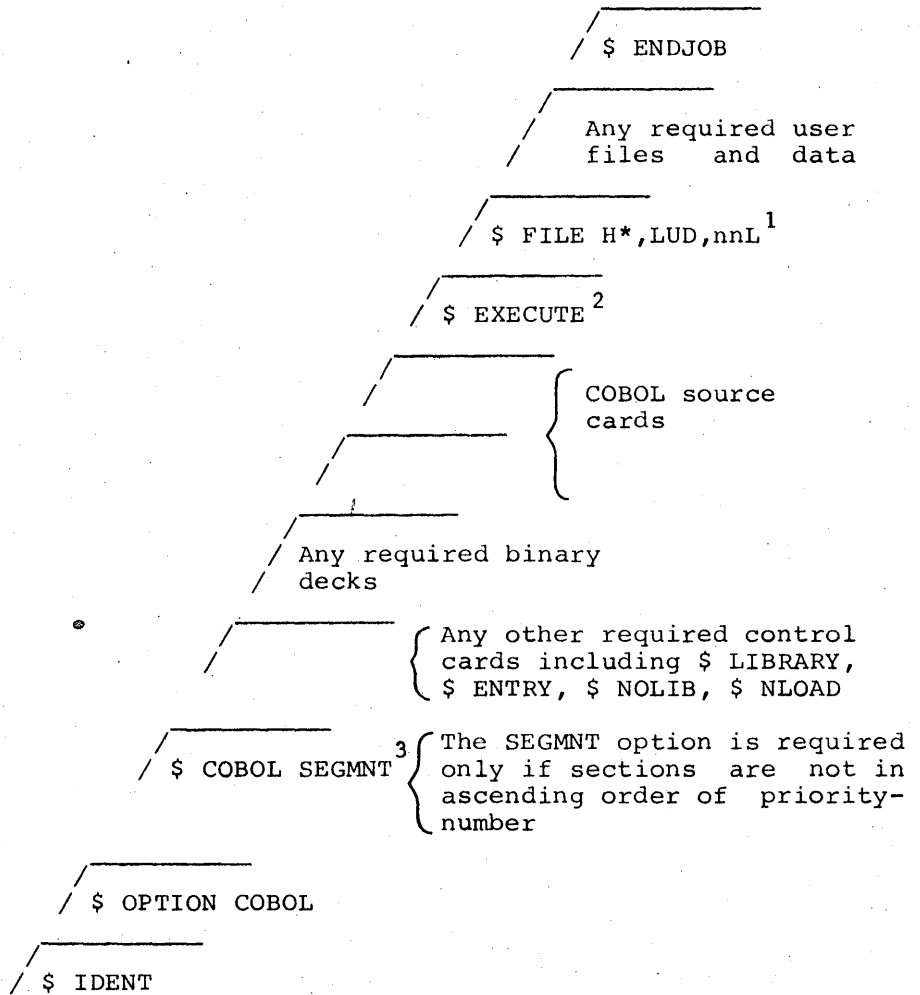




COMPILE USING HIS COPY



COMPILE USING AMERICAN NATIONAL STANDARD SEGMENTATION



<sup>1</sup> LUD is the logical unit designator and nn is the number of sequential links required to store the program segments.

<sup>2</sup> \$ EXECUTE SEG may also be used; this option allocates an H\* file of five sequential links.

<sup>3</sup> If the segmented program is an I-D-S program, the SEGMNT option must be included on the \$ IDS card.



## APPENDIX C

### NONSTANDARD FEATURE FLAGGING

Nonstandard language features can be flagged on the COBOL source program listing by including the FLAGNA option in the operand field of the \$ COBOL card.

If FLAGNA is specified, the compiler inserts the prefix NA immediately preceding the reference number of any source line containing a language feature that is not included in American National Standard COBOL (X3.23-1968). If the option is specified and flagging occurs, the compiler also causes the following message to be printed at the end of the COBOL listing:

```
NA NON ANS COBOL-1968 FEATURE USED.
```

If the FLAGNA option is not specified or if it is specified but no flagging occurs, the preceding message is omitted.

This option has no other effect on the compilation process (i.e., nonstandard source lines are processed even if the option is specified).

The following is a list of nonstandard language elements that are flagged if the FLAGNA option is specified:

- Figurative Constants

UPPER-BOUND(S)

LOWER-BOUND(S)

- HIS COPY Function

{ level indicator } data-name-1 COPY data-name-2 [ FROM LIBRARY ]  
{ level-number }

paragraph-name. COPY library-name [ FROM LIBRARY ]

- Identification Division (None.)

SOURCE-COMPUTER.

[ 6000 [ WITH EIS ] ]  
[ WITH SUPERVISOR CONTROL ]  
[ MEMORY SIZE ... ]  
[ , [integer] hardware-name ]

OBJECT-COMPUTER.

[ 6000 [ WITH EIS ] ]  
[ WITH SUPERVISOR CONTROL ]  
{ ADDRESS ... }  
[ , [integer] hardware-name ]

SPECIAL-NAMES.

[ HMS ]  
[ BLOCK ... ]  
[ COLLATE COMMERCIAL ]  
[ OPTIMIZE { COMPUTATIONAL  
                  COMP } ]  
[ PROCESS ... DEBUG ]  
[ COMPILE PHASE1 ... ]  
[ ELECT SORT ... ]

FILE-CONTROL.

{  
          [ OVERLAY ]  
          [ RENAMING ... ]  
SELECT [ FOR CARDS ]  
          [ FOR LISTING ]  
          [ FOR BLANK COMMON ]  
}

I-O-CONTROL.

[ APPLY ... ]

[ RERUN ... ]

[ SAME [ SORT-MERGE ] AREA ... ]

• Data Division

[ PREPARED FOR computer-name ]

CONSTANT SECTION.

File Descriptions:

[ RECORDING MODE ... ]

[ FILE CONTAINS ... ]

LABEL RECORD(S):      BEGINNING-TAPE-LABEL  
                                 BEGINNING-FILE-LABEL  
                                 ENDING-FILE-LABEL  
                                 ENDING-TAPE-LABEL

[ SEQUENCED ON ... ]

Sort-Merge File Descriptions:

[ FILE CONTAINS ... ]

Data and Report Group Descriptions:

Editing clauses: { ZERO SUPPRESS  
CHECK PROTECT } ...  
FLOAT DOLLAR SIGN }

[CLASS IS ...]

{ PICTURE } IS (symbols J or K in the PICTURE character-string)

[POINT LOCATION IS ...]

[RANGE IS ...]

[SIGNED]

[SIZE IS ...]

{ SOURCE IS [SELECTED] ... }

TYPE IS { OVERFLOW HEADING  
OH  
OVERFLOW FOOTING  
OV }

[USAGE IS ] { COMPUTATIONAL-1  
COMP-1  
COMPUTATIONAL-2  
COMP-2  
COMPUTATIONAL-3  
COMP-3  
COMPUTATIONAL-4  
COMP-4  
DISPLAY-1  
DISPLAY-2 }





INITIATE { ALL }

MERGE ...

MOVE { CORR  
CORRESPONDING } identifier-1 TO (series of destination  
identifiers)

MULTIPLY ... BY (series of multiplier/product identifiers)

MULTIPLY ... BY ... GIVING (series of product identifiers)

SUBTRACT ... GIVING (series of difference identifiers)

TERMINATE { ALL }

WRITE ... ADVANCING { TO TOP OF PAGE }

APPENDIX D

COLLATING SEQUENCES

STANDARD COLLATING SEQUENCE

| Stan.<br>Char. | Stan.<br>Octal | Comm.<br>Octal | Card   | Stan.<br>Char. | Stan.<br>Octal | Comm.<br>Octal | Card   |
|----------------|----------------|----------------|--------|----------------|----------------|----------------|--------|
| 0              | 00             | 60             | 0      |                | 40             | 35             | 11-0   |
| 1              | 01             | 61             | 1      | J              | 41             | 36             | 11-1   |
| 2              | 02             | 62             | 2      | K              | 42             | 37             | 11-2   |
| 3              | 03             | 63             | 3      | L              | 43             | 40             | 11-3   |
| 4              | 04             | 64             | 4      | M              | 44             | 41             | 11-4   |
| 5              | 05             | 65             | 5      | N              | 45             | 42             | 11-5   |
| 6              | 06             | 66             | 6      | O              | 46             | 43             | 11-6   |
| 7              | 07             | 67             | 7      | P              | 47             | 44             | 11-7   |
| 8              | 10             | 70             | 8      | Q              | 50             | 45             | 11-8   |
| 9              | 11             | 71             | 9      | R              | 51             | 46             | 11-9   |
| [              | 12             | 16             | 2-8    | -              | 52             | 10             | 11     |
| #              | 13             | 17             | 3-8    | \$             | 53             | 06             | 11-3-8 |
| @              | 14             | 22             | 4-8    | *              | 54             | 07             | 11-4-8 |
| :              | 15             | 72             | 5-8    | )              | 55             | 03             | 11-5-8 |
| >              | 16             | 20             | 6-8    | ;              | 56             | 76             | 11-6-8 |
| Ignore<br>or ? | 17             | 73             | 7-8    | '              | 57             | 21             | 11-7-8 |
| space          | 20             | 00             | blank  | +              | 60             | 23             | 12-0   |
| A              | 21             | 24             | 12-1   | /              | 61             | 11             | 0-1    |
| B              | 22             | 25             | 12-2   | S              | 62             | 50             | 0-2    |
| C              | 23             | 26             | 12-3   | T              | 63             | 51             | 0-3    |
| D              | 24             | 27             | 12-4   | U              | 64             | 52             | 0-4    |
| E              | 25             | 30             | 12-5   | V              | 65             | 53             | 0-5    |
| F              | 26             | 31             | 12-6   | W              | 66             | 54             | 0-6    |
| G              | 27             | 32             | 12-7   | X              | 67             | 55             | 0-7    |
| H              | 30             | 33             | 12-8   | Y              | 70             | 56             | 0-8    |
| I              | 31             | 34             | 12-9   | Z              | 71             | 57             | 0-9    |
| &              | 32             | 05             | 12     | ←              | 72             | 47             | 0-2-8  |
| .              | 33             | 01             | 12-3-8 | ,              | 73             | 12             | 0-3-8  |
| ]              | 34             | 02             | 12-4-8 | %              | 74             | 13             | 0-4-8  |
| (              | 35             | 15             | 12-5-8 | =              | 75             | 14             | 0-5-8  |
| <              | 36             | 74             | 12-6-8 | "              | 76             | 04             | 0-6-8  |
| \              | 37             | 75             | 12-7-8 | !              | 77             | 77             | 0-7-8  |

Except for the control characters with standard octal codes 17 and 77, the Series 60/6000 printer characters are the same as the graphic symbols in this table.

COMMERCIAL COLLATING SEQUENCE

| <u>Comm. Char.</u> | <u>Comm. Octal</u> | <u>Stan. Octal</u> | <u>Card</u> | <u>Printer</u> | <u>Comm. Char.</u> | <u>Comm. Octal</u> | <u>Stan. Octal</u> | <u>Card</u> | <u>Printer</u> |
|--------------------|--------------------|--------------------|-------------|----------------|--------------------|--------------------|--------------------|-------------|----------------|
| Space              | 00                 | 20                 | blank       | Space          | L                  | 40                 | 43                 | 11-3        | L              |
|                    | 01                 | 33                 | 12-3-8      | .              | M                  | 41                 | 44                 | 11-4        | M              |
| ) or X             | 02                 | 34                 | 12-4-8      | ]              | N                  | 42                 | 45                 | 11-5        | N              |
|                    | 03                 | 55                 | 11-5-8      | )              | O                  | 43                 | 46                 | 11-6        | O              |
|                    | 04                 | 76                 | 0-6-8       | "              | P                  | 44                 | 47                 | 11-7        | P              |
|                    | 05                 | 32                 | 12          | &              | Q                  | 45                 | 50                 | 11-8        | Q              |
| \$                 | 06                 | 53                 | 11-3-8      | \$             | R                  | 46                 | 51                 | 11-9        | R              |
| *                  | 07                 | 54                 | 11-4-8      | *              | ≠                  | 47                 | 72                 | 0-2-8       | ←              |
| -                  | 10                 | 52                 | 11          | -              | S                  | 50                 | 62                 | 0-2         | S              |
| /                  | 11                 | 61                 | 0-1         | /              | T                  | 51                 | 63                 | 0-3         | T              |
| ,                  | 12                 | 73                 | 0-3-8       | ,              | U                  | 52                 | 64                 | 0-4         | U              |
| ( or %             | 13                 | 74                 | 0-4-8       | %              | V                  | 53                 | 65                 | 0-5         | V              |
|                    | 14                 | 75                 | 0-5-8       | =              | W                  | 54                 | 66                 | 0-6         | W              |
|                    | 15                 | 35                 | 12-5-8      | (              | X                  | 55                 | 67                 | 0-7         | X              |
|                    | 16                 | 12                 | 2-8         | [              | Y                  | 56                 | 70                 | 0-8         | Y              |
| # or =             | 17                 | 13                 | 3-8         | #              | Z                  | 57                 | 71                 | 0-9         | Z              |
|                    | 20                 | 16                 | 6-8         | >              | 0                  | 60                 | 00                 | 0           | 0              |
| ' or @             | 21                 | 57                 | 11-7-8      | '              | 1                  | 61                 | 01                 | 1           | 1              |
|                    | 22                 | 14                 | 4-8         | @              | 2                  | 62                 | 02                 | 2           | 2              |
| +                  | 23                 | 60                 | 12-0        | +              | 3                  | 63                 | 03                 | 3           | 3              |
|                    |                    |                    |             |                | 4                  | 64                 | 04                 | 4           | 4              |
| A                  | 24                 | 21                 | 12-1        | A              | 5                  | 65                 | 05                 | 5           | 5              |
| B                  | 25                 | 22                 | 12-2        | B              | 6                  | 66                 | 06                 | 6           | 6              |
| C                  | 26                 | 23                 | 12-3        | C              | 7                  | 67                 | 07                 | 7           | 7              |
| D                  | 27                 | 24                 | 12-4        | D              | 8                  | 70                 | 10                 | 8           | 8              |
| E                  | 30                 | 25                 | 12-5        | E              | 9                  | 71                 | 11                 | 9           | 9              |
| F                  | 31                 | 26                 | 12-6        | F              |                    | 72                 | 15                 | 5-8         | :              |
| G                  | 32                 | 27                 | 12-7        | G              |                    | 73                 | 17                 | 7-8         | ?              |
| H                  | 33                 | 30                 | 12-8        | H              |                    | 74                 | 36                 | 12-6-8      | <              |
| I                  | 34                 | 31                 | 12-9        | I              |                    | 75                 | 37                 | 12-7-8      | \              |
| O                  | 35                 | 40                 | 11-0        | ↑              |                    | 76                 | 56                 | 11-6-8      | ;              |
| J                  | 36                 | 41                 | 11-1        | J              |                    | 77                 | 77                 | 0-7-8       | !              |
| K                  | 37                 | 42                 | 11-2        | K              |                    |                    |                    |             |                |

## APPENDIX E

### COBOL ABORT CODES

The following codes appear on the execution report listing to identify abort error conditions detected during the compilation or execution of a COBOL program. These codes originate from within the compiler or from the COBOL software subroutines. When an abort error condition is detected, the appropriate two-character abort code (C1, C2, CA, CH, etc.) appears in place of the NORMAL TERMINATION portion of the termination message on the execution report listing.

| <u>Code</u> | <u>Abort Reason/Action Required</u>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| C0          | <p>A requested input or output file is not contained in the File Address Table when the COBOL source program is compiled.</p> <ul style="list-style-type: none"><li>• A compiler problem is indicated; notify the Honeywell field representative.</li></ul>                                                                                                                                                                                                                                                                                                                                                                                                                                |
| C1          | <p>The compiler address table (*3 file) is full. This file is assigned by the operating system during source program compilation for use as a temporary working area. Its allocated space is fifteen 3840-word random links.</p> <ul style="list-style-type: none"><li>• Check for misplacement of the \$ FILE *3 card in the COBOL source deck (that is, in the execute activity rather than in the compile activity).</li><li>• Increase the size of the *3 file with a \$ FILE *3 card.</li><li>• If the C1 abort still occurs after the size of the *3 file is increased, recompile the program and assign the *1 (intermediate file) and the G* (GMAP source file) to tape.</li></ul> |
| C2          | <p>The fixed portion of the internal data-name table is filled to the point where the largest 01 record on the overflow file cannot be read back into memory. The number of overflow reads/writes is indicated in the summary report at the end of the source program listing.</p> <ul style="list-style-type: none"><li>• Increase memory size on the \$ LIMITS card by a minimum of 5K.</li></ul>                                                                                                                                                                                                                                                                                        |
| C3          | <p>Invalid internal list structures in the Data Division.</p> <ul style="list-style-type: none"><li>• Possibly caused by invalid level structures for some data items that have been processed by the compiler prior to this point.</li><li>• Check placement of \$ COBOL card (must precede the COBOL program source deck).</li></ul>                                                                                                                                                                                                                                                                                                                                                     |

CodeAbort Reason/Action Required

- C4 Substantive stack build error in Report Writer during source program compilation.
- Compilation problem; notify Honeywell field representative.
- C5 Invalid internal list structures of the Report Writer during source program compilation.
- Compilation problem; notify Honeywell field representative.
- C6 The fixed portion of the Report Table has become full during source program compilation.
- Modify the Report Section (report descriptions) in the source program. See Section VIII, Report Writer.
  - Since the Report Table is fixed length with no overflow capability, an increase in memory size has no effect.
- C7 The fixed and variable portion of the Report Table has become full during source program compilation.
- Modify the Report Section (report descriptions) in the source program. See Section VIII, Report Writer.
  - Since the Report Table is fixed length with no overflow capability, an increase in memory size has no effect.
- C8 Invalid internal list structures of the Report Writer analyzers.
- Compilation problem; notify Honeywell field representative.
- C9 The Report Writer generator cannot build its COBOL lists (internal language).
- Compilation problem; notify Honeywell field representative.
- CA Invalid internal list structures of the Report Writer generators.
- Compilation problem; notify Honeywell field representative.
- NOTE: C4, C5, C8, C9, and CA are generic error codes that are given when the compiler is unable to process statements using the Report Writer feature. Since these aborts may reflect discrepancies within the compiler, the Honeywell field representative should be notified.
- CB Invalid end-of-file mark (not 17 or 23 octal) has been encountered in a COBOL object program. An immediate MME GEBORT is executed.
- Incorrectly formatted input tape. Check the tape for invalid data. When a CB abort occurs, the invalid tape mark will appear in index register zero, the associated file-code will be located in the A-register, and the file control block pointer will be contained in index register 2.
- CC An internal syntax processing error has developed within the compiler.
- Compilation problem; notify Honeywell field representative.

CodeAbort Reason/Action Required

- CD The COPY module, which copies text from a COBOL library, requires more memory.
- Specify additional memory via the \$ LIMITS card, if possible. Refer to Section XIV, Library Facility.
- CE Stack overflow occurred while source program syntax was being processed.
- Compilation problem; notify Honeywell field representative.
- CF The use of a RETURN statement was attempted without having a SORT or MERGE statement in control.
- Examine the source program coding for a CALL, GO TO, PERFORM, or ALTER statement within the output section that references a statement or procedure not contained within the output section.
  - Check for omission of the word SECTION from the output section/procedure label.
- CG An object program has attempted to perform one of the following computations:
- The value zero exponentiated by the value zero.
  - The value zero exponentiated by a negative value.
  - A negative value exponentiated by a nonintegral value.
- CH A MME GESYOT was executed to engage the backdoor file facility and a denial return was received (the return code 001 is given in the lower half of the A-register).
- Compilation problem; notify Honeywell field representative.
- CI A backdoor file facility directive (a \$ FUTIL card) has one of the following errors:
- A missing or invalid field delimiter.
  - A missing operand or DUMP directive.
  - An attempt was made to engage the backdoor file facility and this feature is not included in the system configuration.
- CJ Missing level-numbers have been detected on 25 data-names.
- Check for misspelled or missing Procedure Division header card.
- CK An attempt is being made to open a file that is already open.
- Check the logic flow of the user program.
- CL The internal language analyzers have encountered an unrecoverable error.
- Possibly invalid list structures. Compilation problem; notify Honeywell field representative.

CodeAbort Reason/Action Required

- CR      Input-output data is incomplete for a random file in the COBOL object program.
- All input-output data was not transmitted for a random file in the object program.
- CT      An attempt to load the test monitor dummy link was unsuccessful.
- Compilation problem; notify Honeywell field representative.
- SM      COBOL linkage is not in the stack for an object program.
- The following messages are printed:  
         DESIRED LINKAGE NOT IN PUSHDOWN STACK.  
         PUSHDOWN STACK EXCEEDS LIMIT OF 50.

Transaction Processing Abort Codes

- CX      The contents of the OUTPUT-SIZE field specify a message that is too large to be contained in the data field.
- CZ      No station/terminal is currently attempting to access a direct-access message.

## APPENDIX F

### RESERVED GMAP LOCATION SYMBOLS

#### SYMBOLS USING 'FC'

'FC' is the file-code assigned in the FILE-CONTROL paragraph of the Environment Division:

- 'FC'BUFA - The origin of the secondary buffer area.
- 'FC'BUFF - The origin of the primary buffer area.
- 'FC'CHEK - The first word of the RERUN control packet.
- 'FC'EDOF - The entry point of the AT END coding.
- 'FC'FICB - The location symbol of the GFRC file-code block.
- 'FC'RECD - The origin of the 'process area'.
- 'FC'TABL - The location symbol of the file slave table (mass storage).
- V'FC'EOF - The vector switch for the AT END coding.

#### NONFILE DATA SYMBOLS

(NNNNN is the source alter number.)

- CNNNNN - The first word of an 01 level or 77 data item defined in the Constant Section.
- M00001 - The first word of an optional block of temporary storage for procedure coding.
- M00002 - The first word of a block of temporary storage required for procedure coding.
- M00003 - The first word of a block of temporary storage required for procedure coding.
- NMMMMM - The index-name location symbol.
- WNNNNN - The first word of an 01 level or 77 data item defined in the Working-Storage Section.



## REPORT WRITER SYMBOLS

(NNNNN is a report alter number.)

- DNNNNN - DETAIL report group procedures; the alter number is the first card of the report group.
- INNNNN - INITIATE procedure symbol; the alter number is the first RD card of the report.
- KNNNNN - Controller report procedures; the alter number is the first RD card of the report.
- LNNNNN - Report line process area; the alter number is the first RD card of the report.
- QNNNNN - GENERATE report-name procedures; the alter number is the first RD card of the report.
- RNNNNN - Location symbol of the report control block; the alter number is the first RD card of the report.  
  
Also, all nondetail report group procedures; the alter number is the first card of the report group.
- TNNNNN - TERMINATE procedure symbol; the alter number is the first RD card of the report.

## PROCEDURE DIVISION SYMBOLS

(NNNNN is a procedure alter number.)

- C.XXXX - The program entrance symbol for object runs with any \$ entry; XXXX represents the first four characters of the program-name as specified in the Identification Division.
- C.LDIN - The program entrance symbol from the General Loader.
- E00001 - Index register save area (local symbol).
- E00002 - End of program (local symbol).
- E00003 - Undefined location for missing ACTUAL KEY.
- ONNNNN - The first word of an OCCURS...DEPENDING ON control packet; the alter number is that of an 01 level card preceding the elementary item with this clause.
- PNNNNN - Paragraph-name; the alter number is that of the card defining the paragraph-name.
- SNNNNN - Section-name; the alter number is that of the card defining the section-name.
- UNNNNN - USE procedure entry; the alter number is that of the card defining the section-name.
- VNNNNN - Switch vector; the alter number is that of the card defining the GO TO statement.
- XNNNNN - EXIT mechanism symbol; the alter number is that of the concluding source card for the paragraph that precedes the exit point.

- XXXXXX - The CALL program entrance symbol, which is the program-name specified in the Identification Division.
- ZNNNNN - The first word of an ELECT SORT OPTIONS coding packet.

UTILITY SYMBOLS

(NNNNN is a generated number.)

- ANNNNN - Any symbol required for Procedure Division analyzer action.
- BNNNNN - Any subscript calculating subroutine location or temporary storage location.
- G00000 - The location symbol for the base post link for execution with the test monitor, where the value of link is .XX (XX represents the first two characters of program-name).
- G00001 - The location symbol for the Transaction Processing Applications Program (TPAP) identifier.
- GNNNNN - Any symbol required for procedure coding.
- HNNNNN - Any symbol required for procedure coding.
- ZINBUF - The location symbol for the Transaction Processing input buffer.
- ZOTHDR - The location symbol for the Transaction Processing output header.
- ZOUTB1 - The location symbol for the Transaction Processing output buffer.
- ZOUTB2 - The location symbol for the Transaction Processing output buffer.
- .CTABL - Procedure-names table (for segments).
- .CTMP0 - Temporary storage.
- .CTMP1 - Temporary storage.



## APPENDIX G

### COMPILER LIMITATIONS

The COBOL-68 compiler has several operational limitations that are either not documented in previous editions of the Honeywell user's manuals, or are discussed in obscure locations in the manuals. These restrictions are therefore recorded in this appendix.

#### SOURCE PROGRAM SIZE LIMITATION

The number of lines in a COBOL-68 source program may not exceed 32,767.

#### MISPLACED ERROR MESSAGES

An error message associated with an incorrect COBOL statement may appear next to an inappropriate alter number if the error is caused by any of the following conditions:

1. The error is contained in the final word on a line.
2. Text that should follow the final word on a line is omitted.
3. The terminating period on a line is omitted.

The alter number associated with the error message will be that of the first noncomment line following the statement containing the error.

If a comment line is present and is one of the special comment lines containing LSTOF, the error message is suppressed. Therefore, any diagnostic messages that are printed following a line containing LSTOF and before the next line containing LSTON are not printed; however, they are counted in the respective summary report totals.

## IDENTIFICATION DIVISION LIMITATIONS

In the PROGRAM-ID paragraph, the first four characters of program-name must not be LDIN. If the program-name exceeds six characters, it will be truncated, and any invalid characters will be replaced by a period. If a program is to be utilized within the Transaction Processing System, the first three characters of program-name are used as the TPAP identifier and must be unique within the Transaction Processing System. If a program is to be loaded into the same overlay as other COBOL programs, the first four characters of each program-name must be unique within that overlay.

In the REMARKS paragraph, all lines of the comment-entry are restricted to Area B of the reference format.

## ENVIRONMENT DIVISION LIMITATIONS

All programs in the same run unit should be compiled in either the EIS mode or the NEIS mode.

In the BLOCK option of the SPECIAL-NAMES paragraph, a maximum of 63 Labeled Common storage blocks are permitted in a given program.

## DATA DIVISION LIMITATIONS

The maximum size of an 01 level record description entry is 99,999 words.

### Data Division Compilation

When the Data Division is being compiled, data entries are stored in a unified data table. Approximately 600 entries can be contained in the available memory area if the standard compilation memory limit (32K) is used. The compiler includes provisions for dynamically expanding the size of the data table. When the data table, as initially allocated, is full, the compiler attempts to obtain additional memory in 4096 (4K) word increments via the MME GEMORE function. If successful, the data table is increased by the amount of memory specified and the source program processing continues. If three denials are received, the data table enters an overflow status and the following discussion is applicable.

If the available memory area is not sufficient to contain the entire data table, portions will be written to the overflow (\*3) file. When the data table is in an overflow condition, spurious error messages and/or incomplete object program coding may be produced if one of the following types of statements is encountered:

1. An ADD, MOVE, or SUBTRACT statement that contains a CORRESPONDING phrase.
2. A conditional statement that contains a subscripted data item in which the subscript is a data-name.
3. A SEARCH ALL statement or a SEARCH statement that specifies the VARYING phrase.

4. A REDEFINES clause describing large data record descriptions that are defined in the File Section.

If spurious error messages are received and the summary report listing indicates the presence of overflow reads or writes, additional memory should be allocated via a \$ LIMITS card until the number of overflow reads and writes is reduced to zero. This causes the unified data table to be entirely memory contained and most programs affected by overflow problems will then compile correctly.



## Report Group Entry

A spurious warning message that indicates a requirement for fractional truncation is produced for a report group entry that specifies both a PICTURE clause having a trailing scaling character 'P' and the VALUE clause. The program executes correctly.

Variable-length data items may not be defined in the Report Section of the Data Division.

## Table Handling

The maximum size of a single-dimension array is 262,143 characters.

The maximum size of a multiple-dimension array is 524,287 characters.

A literal subscript calculation that yields an effective address equal to or greater than 262,143 may produce unpredictable results.

The maximum value that can be specified in an OCCURS clause is 32,767.

## PICTURE Clause

The maximum size of a data item that can be specified with a PICTURE clause is 131,071 characters.

## REDEFINES Clause

When the REDEFINES clause is specified in the Working-Storage Section and more than fifty noncontiguous data items (level 77) are defined, the REDEFINES clause and the item it redefines must be included in the same group of fifty items; that is, in the first fifty level 77 items, or the second fifty level 77 items, etc.

## VALUE Clause

When Format 2 of the VALUE clause is specified (a level 88 condition-name entry), no more than 24 ranges of values (THRU phrases) may be used.

If a data item has an ending character position within a record that is equal to or greater than 262,143, the data item must not be referenced by a Procedure Division statement and must not include a VALUE clause in its data description.



## PROCEDURE DIVISION LIMITATIONS

### Segmentation

The first paragraph-name in a section within a segmented program must be contiguous to the section-name; that is, no intervening blank lines, comment lines, NOTE sentences, and/or EJECT lines are allowed.

### COPY Statement

A blank line must not immediately follow a COPY statement. A maximum of 99 COPY statements are permitted within a given COBOL program.

### American National Standard COPY WITH CCOMDK Option

An erroneous data structure may be produced when the following conditions exist:

1. Both the LIBCOPY option and the CCOMDK option are specified in the variable field on the \$ COBOL card, and
2. A COPY statement with a terminating period in column 72 is specified.

These conditions cause the source statement following the COPY statement that has a period in column 72 to be treated as a comment by incorrectly inserting an asterisk (\*) in column 7 of the entry.

### IF Conditional Statements

Undefined GMAP symbols may be produced when the following conditions exist:

1. A single IF conditional statement contains references to more than 17 conditional variables (level 88 items) which are connected by either an AND or an OR logical operator, and
2. Each conditional variable is defined using the THRU phrase of the VALUE clause.

A compiler abnormal termination may result when the following conditions exist:

1. More than 13 nested IF conditional statements are specified, and
2. Each IF conditional statement references a conditional variable (level 88 item) which is defined using the THRU phrase of the VALUE clause.

### MOVE Statement

A spurious warning message that indicates a requirement for fractional truncation is produced when an integer data item having a trailing scaling character 'P' is the subject of a MOVE statement. The program executes correctly.

### NOTE Paragraph

If a NOTE paragraph contains more than one sentence, the appearance of the reserved word COPY in the second or subsequent sentences will produce the following extraneous error message during compilation:

```
ER ILLEGAL CHARACTER IN COLUMN 7 -- DELETED CHARACTER
```

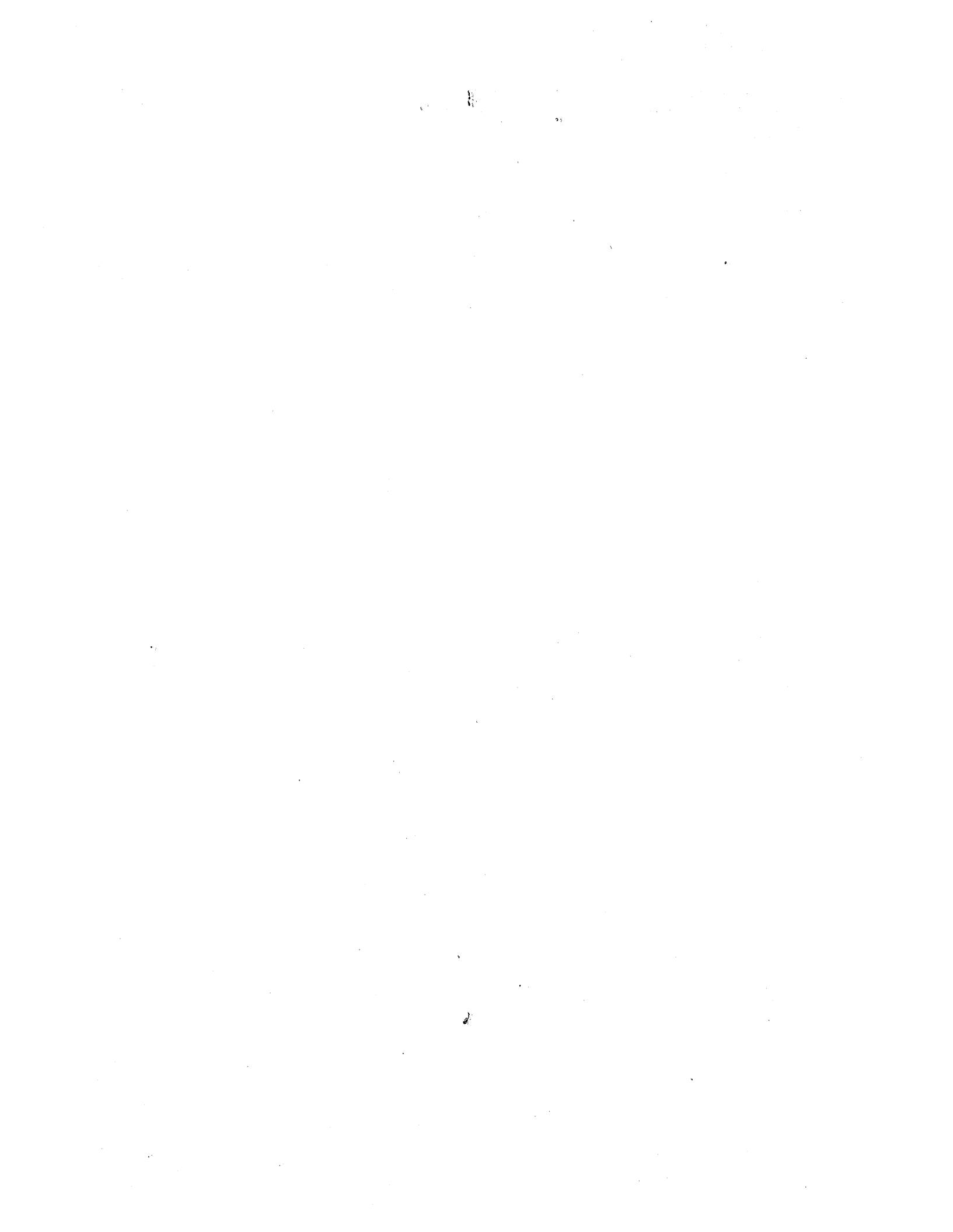
The compiled program will execute correctly.

### OPEN Statement

If the total number of files concurrently open in a COBOL program approaches 25-30, the amount of available space in the peripheral allocation table (PAT) could become exhausted and an abnormal GCOS termination may occur. This is an operating system limitation.

### PERFORM Statement

When a PERFORM... VARYING... AFTER... FROM... BY statement is compiled in the non-EIS mode, spurious GMAP 'M' flags may be produced if radix conversion is required for any operand specified in the AFTER, FROM, or BY phrases. The resultant object program will execute correctly.



If an incomplete condition is specified within a PERFORM statement which is itself the imperative portion of an IF conditional statement, no error message is produced.

Example: IF A = B PERFORM C VARYING D FROM 1 BY 1 UNTIL > E.

In the above example, no error flag will be generated for the incomplete condition '> E'. Coding is produced as if the condition were written as 'A > E'.

#### READ Statement

If a READ...AT END statement is specified as a part of the imperative-statement for another READ...AT END statement, no error message is printed and undefined GMAP symbols may appear on the source program listing.

#### SORT-MERGE Statements

The following comment does not represent a compiler limitation but is documented in this appendix because the implicit nature of the SORT input and output procedures and the MERGE output procedure is often misunderstood.

The input procedure and output procedure of a SORT statement, and the output procedure of a MERGE statement, are considered to be PERFORM mechanisms; they must conform to the rules for entering and exiting as specified for the PERFORM statement. If these rules are not followed, a GCOS or SORT error message will be printed and an abnormal termination will occur.

#### USE Statement

If Format 2 of the USE statement is specified, the label procedures must not execute any ACCEPT FROM GIN, CLOSE, DISPLAY, DISPLAY UPON SYSOUT, OPEN, READ, SEEK, or WRITE statements.

#### WRITE Statement

The ADVANCING phrase in Format 1 of the WRITE statement must not be specified for a file that contains an OCCURS...DEPENDING clause.



INDEX

|                    |  |      |
|--------------------|--|------|
| \$ COBOL           |  |      |
| \$ COBOL card      |  | B-1  |
| \$ ETC             |  |      |
| \$ ETC             |  | B-3  |
| \$ EXECUTE         |  |      |
| \$ EXECUTE         |  | 15-8 |
| \$ EXECUTE         |  | 6-7  |
| \$ LIMITS          |  |      |
| \$ LIMITS          |  | 16-3 |
| \$ LIMITS          |  | B-3  |
| \$ LINK            |  |      |
| \$ LINK            |  | 15-6 |
| \$ NTAPE           |  |      |
| \$ NTAPE           |  | 9-16 |
| \$ TAPE            |  |      |
| \$ TAPE            |  | 9-16 |
| 'FC'               |  |      |
| SYMBOLS USING 'FC' |  | F-1  |
| .CMEQK             |  |      |
| .CMEQK             |  | 9-24 |
| .CSEQK             |  |      |
| .CSEQK             |  | 9-8  |
| 66                 |  |      |
| Level-number 66    |  | 4-4  |
| level-number 66    |  | 4-13 |
| level-number 66    |  | 2-4  |
| 77                 |  |      |
| Level-number 77    |  | 4-4  |
| level 77 entries   |  | 17-3 |
| level-number 77    |  | 2-3  |
| 88                 |  |      |
| Level-number 88    |  | 4-4  |
| level-number 88    |  | 4-1  |
| level-number 88    |  | 2-4  |

|                                                      |       |
|------------------------------------------------------|-------|
| ABORT CODES                                          |       |
| ABORT CODES                                          | E-1   |
| Transaction Processing Abort Codes                   | E-4   |
| ABORTS                                               |       |
| Compilation Aborts                                   | 16-8  |
| ACCEPT                                               |       |
| ACCEPT STATEMENTS                                    | 6-1   |
| ACCEPT MESSAGE                                       |       |
| ACCEPT MESSAGE                                       | 7-5   |
| ACCESS MODE                                          |       |
| ACCESS MODE Phrase                                   | 3-7   |
| ACTUAL KEY                                           |       |
| ACTUAL KEY                                           | 5-20  |
| ACTUAL KEY                                           | 5-2   |
| ACTUAL KEY Phrase                                    | 3-8   |
| ADD                                                  |       |
| ADD Statement                                        | 11-7  |
| ADVANCING                                            |       |
| ADVANCING PHRASE                                     | 5-23  |
| ALIGNMENT                                            |       |
| alignment                                            | 4-14  |
| ALLOCATION                                           |       |
| Resource Allocation                                  | 16-5  |
| ALPHABETIC                                           |       |
| ALPHABETIC                                           | 12-3  |
| ALPHABETIC                                           | 17-4  |
| Alphabetic                                           | 10-2  |
| ALPHANUMERIC                                         |       |
| ALPHANUMERIC                                         | 17-4  |
| Alphanumeric                                         | 10-2  |
| ALPHANUMERIC EDITED                                  |       |
| Alphanumeric Edited                                  | 10-2  |
| ALTER                                                |       |
| ALTER                                                | 9-10  |
| ALTER                                                | 15-5  |
| ALTER                                                | 9-26  |
| AMERICAN NATIONAL STANDARD                           |       |
| AMERICAN NATIONAL STANDARD COPY                      | 14-10 |
| AMERICAN NATIONAL STANDARD COPY WITH CCOMDK          | 14-12 |
| AMERICAN NATIONAL STANDARD COPY WITH COMDK           | 14-12 |
| AND                                                  |       |
| AND                                                  | 12-5  |
| APPLICATIONS                                         |       |
| TRANSACTION PROCESSING APPLICATIONS PROGRAMS (TPAPS) | 7-3   |
| Transaction Processing Applications Program Example  | 7-9   |
| APPLY                                                |       |
| APPLY PHRASE                                         | 3-8   |
| APPLY PROCESS AREA phrase                            | 16-9  |

|                                           |       |
|-------------------------------------------|-------|
| AREA                                      |       |
| SAME AREA                                 | 5-14  |
| SAME AREA Phrase                          | 3-10  |
| SAME AREA phrase                          | 16-9  |
| SAME RECORD AREA                          | 5-15  |
| SAME RECORD AREA Phrase                   | 3-10  |
| SAME RECORD AREA phrase                   | 16-10 |
| SAME SORT or SORT-MERGE AREA Phrase       | 3-11  |
| Buffer Areas                              | 5-14  |
| FILE PROCESSING AREAS                     | 5-14  |
| Labeled Common areas                      | 9-19  |
| Labeled Common areas                      | 9-29  |
| Record Areas                              | 5-14  |
| Sort Areas and Sort-Merge Areas           | 5-16  |
| ARITHMETIC                                |       |
| ARITHMETIC STATEMENTS                     | 11-6  |
| arithmetic operators                      | 11-2  |
| Multiple Results in Arithmetic Statements | 11-10 |
| ASCENDING                                 |       |
| ASCENDING option                          | 9-8   |
| ASCENDING option                          | 9-23  |
| ASSIGN                                    |       |
| ASSIGN Phrase                             | 3-4   |
| ASSIGN phrase                             | 9-4   |
| AT END                                    |       |
| AT END                                    | 9-12  |
| AT END                                    | 9-27  |
| AT END                                    | 5-4   |
| AT END PHRASE                             | 5-21  |
| AT END phrase                             | 13-8  |
| BACKDOOR FILE                             |       |
| backdoor file                             | 16-10 |
| BITS                                      |       |
| BITS option                               | 17-7  |
| BLANK COMMON                              |       |
| FOR BLANK COMMON Phrase                   | 3-7   |
| BLANK WHEN ZERO                           |       |
| BLANK WHEN ZERO Clause                    | 4-1   |
| BLOCK CONTAINS                            |       |
| BLOCK CONTAINS Clause                     | 3-13  |
| BLOCK SERIAL NUMBER                       |       |
| BLOCK SERIAL NUMBER Phrase                | 3-9   |
| BLOCKING FACTOR                           |       |
| blocking factor                           | 3-14  |
| BREAK                                     |       |
| control break                             | 8-7   |
| Overflow Breaks                           | 8-8   |
| Page Breaks                               | 8-8   |



|                                             |  |       |
|---------------------------------------------|--|-------|
| BUFFER                                      |  |       |
| Buffer Areas                                |  | 5-14  |
| CALCULATIONS                                |  |       |
| POST-SLEW CALCULATIONS                      |  | 8-15  |
| PRE-SLEW CALCULATIONS                       |  | 8-14  |
| CALL                                        |  |       |
| CALL STATEMENT                              |  | 15-14 |
| CARD                                        |  |       |
| \$ COBOL card                               |  | B-1   |
| File Control Cards                          |  | 5-10  |
| CCOMDK                                      |  |       |
| AMERICAN NATIONAL STANDARD COPY WITH CCOMDK |  | 14-12 |
| HIS COPY WITH CCOMDK                        |  | 14-5  |
| CHARACTER-STRING                            |  |       |
| character-string                            |  | 4-8   |
| CHECK                                       |  |       |
| label check                                 |  | 5-27  |
| CHECK PROTECT                               |  |       |
| CHECK PROTECT                               |  | 17-6  |
| CHECKPOINT                                  |  |       |
| checkpoint dump                             |  | 3-10  |
| CLASS                                       |  |       |
| CLASS Clause                                |  | 17-4  |
| CLASS CONDITION                             |  | 12-3  |
| CLASSIFICATION                              |  |       |
| Segment Classification                      |  | 15-2  |
| CLAUSES                                     |  |       |
| Editing Clauses                             |  | 17-5  |
| CLOSE                                       |  |       |
| CLOSE Statement                             |  | 5-25  |
| STANDARD CLOSE FILE                         |  | 5-26  |
| STANDARD CLOSE REEL                         |  | 5-26  |
| Closed Status                               |  | 5-2   |
| COBOL                                       |  |       |
| COBOL INPUT STATEMENT PROCESSING            |  | 7-5   |
| COBOL OUTPUT STATEMENT PROCESSING           |  | 7-7   |
| COBOL USER'S GUIDE ORGANIZATION             |  | 1-2   |
| ORDER OF COBOL SOURCE PROGRAM               |  | A-1   |
| CODE                                        |  |       |
| CODE clause                                 |  | 8-9   |
| COLLATING SEQUENCE                          |  |       |
| COMMERCIAL COLLATING SEQUENCE               |  | D-2   |
| commercial collating sequence               |  | 2-24  |
| commercial collating sequence               |  | 4-18  |
| STANDARD COLLATING SEQUENCE                 |  | D-1   |
| collating sequences                         |  | 12-2  |
| COLLATION                                   |  |       |
| collation files                             |  | 9-15  |

|                                                           |       |
|-----------------------------------------------------------|-------|
| COMDK                                                     |       |
| AMERICAN NATIONAL STANDARD COPY WITH COMDK                | 14-12 |
| HIS COPY WITH COMDK                                       | 14-5  |
| COMMERCIAL                                                |       |
| COMMERCIAL COLLATING SEQUENCE                             | D-2   |
| commercial collating sequence                             | 4-18  |
| commercial collating sequence                             | 2-24  |
| COMMUNICATION-DEVICE                                      |       |
| COMMUNICATION-DEVICE                                      | 6-4   |
| COMMUNICATION-DEVICE                                      | 7-7   |
| COMMUNICATION-DEVICE                                      | 7-5   |
| COMMUNICATIONS                                            |       |
| Procedure Division Communications                         | 15-13 |
| COMP-3 PK SYNC                                            |       |
| COMP-3 PK SYNC                                            | 4-18  |
| COMPARISONS                                               |       |
| Comparisons Involving Index-Names and/or Index Data Items | 13-10 |
| COMPILATION                                               |       |
| COMPILATION TECHNIQUES                                    | 16-2  |
| Compilation Aborts                                        | 16-8  |
| COMPILE PHASE1 ONLY                                       |       |
| COMPILE PHASE1 ONLY option                                | 16-2  |
| COMPRESSED                                                |       |
| COMPRESSED DECKS                                          | 16-14 |
| COMPUTATION                                               |       |
| METHODS OF COMPUTATION                                    | 11-1  |
| COMPUTATIONAL                                             |       |
| COMPUTATIONAL                                             | 16-13 |
| COMPUTATIONAL                                             | 15-16 |
| COMPUTATIONAL                                             | 4-17  |
| COMPUTATIONAL Data Items                                  | 2-27  |
| COMPUTATIONAL Item Formats                                | 2-25  |
| COMPUTATIONAL items                                       | 4-15  |
| COMPUTATIONAL-1                                           |       |
| COMPUTATIONAL-1 Data Items                                | 2-30  |
| COMPUTATIONAL-2                                           |       |
| COMPUTATIONAL-2 Data Items                                | 2-31  |
| COMPUTATIONAL-3                                           |       |
| COMPUTATIONAL-3 Data Items                                | 2-31  |
| COMPUTATIONAL-3 PACKED SYNCHRONIZED Data Items            | 2-32  |
| COMPUTE                                                   |       |
| COMPUTE Statement                                         | 11-7  |
| CONDITION                                                 |       |
| CLASS CONDITION                                           | 12-3  |
| CONDITION-NAME CONDITION                                  | 12-4  |
| overflow condition                                        | 8-9   |
| RELATION CONDITION                                        | 12-1  |
| SIGN CONDITION                                            | 12-3  |
| SWITCH-STATUS CONDITION                                   | 12-5  |

|                                             |  |       |
|---------------------------------------------|--|-------|
| CONDITION-NAME                              |  |       |
| CONDITION-NAME CONDITION                    |  | 12-4  |
| Condition-Name Entries                      |  | 2-4   |
| Condition-Name Entry                        |  | 4-1   |
| condition-name entries                      |  | 4-20  |
| CONDITIONAL                                 |  |       |
| CONDITIONAL PROCEDURES                      |  | 12-1  |
| CONDITIONAL STATEMENTS                      |  | 17-10 |
| CONDITIONAL VARIABLE                        |  |       |
| conditional variable                        |  | 12-4  |
| conditional variable                        |  | 4-20  |
| conditional variable                        |  | 4-2   |
| CONDITIONS                                  |  |       |
| Abbreviated Combined Relation Conditions    |  | 12-8  |
| Compound Conditions                         |  | 12-5  |
| Evaluation Rules for Conditions             |  | 12-14 |
| Simple Conditions                           |  | 12-1  |
| CONSOLE                                     |  |       |
| CONSOLE                                     |  | 6-6   |
| CONSOLE INTERACTION                         |  |       |
| console interaction                         |  | 6-6   |
| CONSTANT                                    |  |       |
| CONSTANT SECTION                            |  | 17-2  |
| Constant Records                            |  | 17-3  |
| Noncontiguous Constant Storage              |  | 17-2  |
| Tables of Constants                         |  | 17-3  |
| CONSTRAINTS                                 |  |       |
| Linked Overlay Environment Constraints      |  | 15-16 |
| REPORT WRITER TABLE CONSTRAINTS             |  | 8-16  |
| CONTROL                                     |  |       |
| Control Data Items                          |  | 8-7   |
| control break                               |  | 8-7   |
| File Control Cards                          |  | 5-10  |
| Report Control                              |  | 8-2   |
| Segmentation Control                        |  | 15-3  |
| slew control                                |  | 6-4   |
| slew control                                |  | 5-23  |
| Transfer of Control                         |  | 15-5  |
| CONVERSION                                  |  |       |
| span conversion                             |  | 2-28  |
| COPY                                        |  |       |
| AMERICAN NATIONAL STANDARD COPY             |  | 14-10 |
| AMERICAN NATIONAL STANDARD COPY WITH CCOMDK |  | 14-12 |
| AMERICAN NATIONAL STANDARD COPY WITH COMDK  |  | 14-12 |
| COPY Clause                                 |  | 4-2   |
| COPY FUNCTIONS                              |  | 14-1  |
| HIS COPY                                    |  | 14-1  |
| HIS COPY WITH CCOMDK                        |  | 14-5  |
| HIS COPY WITH COMDK                         |  | 14-5  |
| CORRELATION                                 |  |       |
| SOURCE/SUM Correlation                      |  | 8-13  |

|                                                |  |       |
|------------------------------------------------|--|-------|
| CORRESPONDING                                  |  |       |
| CORRESPONDING Option                           |  | 11-5  |
| COUNTER                                        |  |       |
| Line Counter                                   |  | 8-9   |
| Page Counter                                   |  | 8-10  |
| SUM Counter Manipulation                       |  | 8-10  |
| CROSS-REFERENCE                                |  |       |
| CROSS-REFERENCE FACILITY                       |  | 16-15 |
| CROSSFOOTING                                   |  |       |
| CROSSFOOTING                                   |  | 8-12  |
| DAC                                            |  |       |
| Direct-Access (DAC) Mode Processing            |  | 7-8   |
| DATA                                           |  |       |
| DATA STRUCTURES                                |  | 2-1   |
| hierarchy of data                              |  | 4-3   |
| NONFILE DATA SYMBOLS                           |  | F-1   |
| positioning of data                            |  | 4-3   |
| DATA DESCRIPTION                               |  |       |
| DATA DESCRIPTION ENTRIES                       |  | 2-22  |
| Data Description Techniques                    |  | 16-13 |
| DATA FORMATS                                   |  |       |
| External Data Formats                          |  | 2-11  |
| Internal Data Formats                          |  | 2-16  |
| DATA ITEMS                                     |  |       |
| COMPUTATIONAL Data Items                       |  | 2-27  |
| COMPUTATIONAL-1 Data Items                     |  | 2-30  |
| COMPUTATIONAL-2 Data Items                     |  | 2-31  |
| COMPUTATIONAL-3 Data Items                     |  | 2-31  |
| COMPUTATIONAL-3 PACKED SYNCHRONIZED Data Items |  | 2-32  |
| Control Data Items                             |  | 8-7   |
| Tables of Data Items                           |  | 2-6   |
| DATA MANIPULATION                              |  |       |
| Data Manipulation Techniques                   |  | 16-12 |
| DATA MOVEMENT                                  |  |       |
| DATA MOVEMENT                                  |  | 10-1  |
| DATA RECORDS                                   |  |       |
| DATA RECORD(S) Clause                          |  | 3-16  |
| DATA TRANSMISSION                              |  |       |
| DATA TRANSMISSION PROGRAM EXAMPLE              |  | 6-9   |
| DATA TRANSMISSION TECHNIQUES                   |  | 6-3   |
| LOW-VOLUME DATA TRANSMISSION                   |  | 6-1   |
| DATA-NAME                                      |  |       |
| DATA-NAME OPTION                               |  | 3-18  |
| DATE                                           |  |       |
| Date and Time                                  |  | 6-5   |
| DEBUG                                          |  |       |
| DEBUG STATEMENTS                               |  | 16-1  |
| DECIMAL POINT                                  |  |       |
| assumed decimal point                          |  | 17-7  |

|                                     |  |       |
|-------------------------------------|--|-------|
| DECK SETUPS                         |  |       |
| DECK SETUPS                         |  | B-1   |
| DECKS                               |  |       |
| COMPRESSED DECKS                    |  | 16-14 |
| DECLARATIONS                        |  |       |
| Merge Key Declarations              |  | 9-22  |
| Sort Key Declarations               |  | 9-5   |
| DEPENDING                           |  |       |
| DEPENDING phrase                    |  | 4-5   |
| DESCENDING                          |  |       |
| DESCENDING option                   |  | 9-8   |
| DESCENDING option                   |  | 9-23  |
| DEVICE                              |  |       |
| device independence                 |  | 5-10  |
| Peripheral Devices                  |  | 5-12  |
| Remote Devices                      |  | 6-4   |
| DIRECT-ACCESS                       |  |       |
| Direct-Access (DAC) Mode Processing |  | 7-8   |
| DISPLAY                             |  |       |
| DISPLAY                             |  | 4-17  |
| DISPLAY                             |  | 7-7   |
| DISPLAY Item Formats                |  | 2-24  |
| DISPLAY STATEMENTS                  |  | 6-2   |
| DISPLAY-1                           |  |       |
| DISPLAY-1                           |  | 2-24  |
| DISPLAY-2                           |  |       |
| DISPLAY-2                           |  | 2-24  |
| DIVIDE                              |  |       |
| DIVIDE Statement                    |  | 11-8  |
| DOWN BY                             |  |       |
| DOWN BY                             |  | 13-9  |
| DUMP                                |  |       |
| checkpoint dump                     |  | 3-10  |
| DUMP option                         |  | 16-9  |
| DUPLICATE                           |  |       |
| duplicate text                      |  | 4-2   |
| EDITING                             |  |       |
| EDITING RULES                       |  | 4-10  |
| Editing Clauses                     |  | 17-5  |
| ELECT                               |  |       |
| ELECT                               |  | 9-15  |
| ELECT                               |  | 9-11  |
| ELECT phrase                        |  | 9-31  |
| ELECT phrase                        |  | 9-19  |

|                                      |       |
|--------------------------------------|-------|
| ELEMENTARY ITEM                      |       |
| ELEMENTARY ITEM DESCRIPTION ENTRIES  | 4-1   |
| elementary item                      | 4-13  |
| elementary item                      | 4-8   |
| elementary item                      | 8-4   |
| ELEMENTARY ITEMS                     | 16-5  |
| GROUPS OF ELEMENTARY ITEMS           | 4-20  |
| Group Items and Elementary Items     | 2-1   |
| Noncontiguous Elementary Items       | 2-3   |
| <br>                                 |       |
| ELEMENTARY MOVE                      |       |
| elementary MOVE                      | 10-1  |
| <br>                                 |       |
| ELEMENTS                             |       |
| OBSOLETE LANGUAGE ELEMENTS           | 17-1  |
| <br>                                 |       |
| END-OF-MESSAGE                       |       |
| END-OF-MESSAGE                       | 7-7   |
| <br>                                 |       |
| END-OF-SEGMENT                       |       |
| END-OF-SEGMENT                       | 7-7   |
| <br>                                 |       |
| END-OF-TRANSACTION                   |       |
| END-OF-TRANSACTION                   | 7-7   |
| <br>                                 |       |
| ENTRIES                              |       |
| Condition-Name Entries               | 2-4   |
| condition-name entries               | 4-20  |
| DATA DESCRIPTION ENTRIES             | 2-22  |
| ELEMENTARY ITEM DESCRIPTION ENTRIES  | 4-1   |
| FILE DESCRIPTION ENTRIES             | 3-12  |
| GROUP ENTRIES                        | 8-21  |
| Level-Number/Data-Name Entries       | 4-3   |
| level 77 entries                     | 17-3  |
| RD Entries                           | 8-5   |
| REDEFINES Entries                    | 2-3   |
| RENAMES Entries                      | 2-4   |
| REPORT GROUP ENTRIES                 | 8-20  |
| Report Group Entries                 | 8-5   |
| SORT-MERGE FILE DESCRIPTION ENTRIES  | 3-13  |
| SOURCE ENTRIES                       | 8-21  |
| SUM ENTRIES                          | 8-23  |
| VALUE ENTRIES                        | 8-24  |
| <br>                                 |       |
| ENTRY                                |       |
| Condition-Name Entry                 | 4-1   |
| <br>                                 |       |
| ENTRY POINT                          |       |
| ENTRY POINT PHRASE                   | 15-15 |
| <br>                                 |       |
| EQUAL KEY                            |       |
| MERGE EQUAL KEY RECORD PROCESSING    | 9-25  |
| Merge Equal Key Procedures           | 9-23  |
| SORT EQUAL KEY RECORD PROCESSING     | 9-9   |
| Sort Equal Key Procedures            | 9-8   |
| <br>                                 |       |
| EQUALS                               |       |
| EQUALS                               | 13-7  |
| <br>                                 |       |
| ERROR PROCEDURE                      |       |
| ERROR PROCEDURE PHRASE               | 5-27  |
| <br>                                 |       |
| EXAMINE                              |       |
| REPLACING PHRASE (EXAMINE STATEMENT) | 10-8  |
| TALLYING PHRASE (EXAMINE STATEMENT)  | 11-12 |

|                                                     |       |
|-----------------------------------------------------|-------|
| EXAMPLE                                             |       |
| DATA TRANSMISSION PROGRAM EXAMPLE                   | 6-9   |
| Multiple Module Program Example                     | 15-17 |
| REPORT WRITER PROGRAM EXAMPLE                       | 8-27  |
| Transaction Processing Applications Program Example | 7-9   |
| EXAMPLES OF MOVE CORRESPONDING STATEMENTS           | 10-7  |
| EXAMPLES OF MOVE STATEMENTS                         | 10-3  |
| FILE PROCESSING EXAMPLES                            | 5-29  |
| Merge Examples                                      | 9-29  |
| SEARCH and SET Statement Examples                   | 13-10 |
| Sort Examples                                       | 9-20  |
| EXIT                                                |       |
| EXIT                                                | 15-14 |
| EXIT STATEMENT                                      | 15-15 |
| EXPONENTIATION                                      |       |
| exponentiation                                      | 11-3  |
| FIGURATIVE CONSTANTS                                |       |
| figurative constants                                | 10-1  |
| FILE                                                |       |
| *3 File                                             | 16-6  |
| File Control Cards                                  | 5-10  |
| File Utilization                                    | 16-7  |
| STANDARD CLOSE FILE                                 | 5-26  |
| The Merge File                                      | 9-22  |
| The Sort File                                       | 9-4   |
| FILE CONTAINS                                       |       |
| FILE CONTAINS Clause                                | 17-6  |
| FILE DESCRIPTION                                    |       |
| FILE DESCRIPTION ENTRIES                            | 3-12  |
| SORT-MERGE FILE DESCRIPTION ENTRIES                 | 3-13  |
| FILE DESCRIPTIONS                                   | 3-1   |
| FILE PROCESSING                                     |       |
| FILE PROCESSING AREAS                               | 5-14  |
| FILE PROCESSING CONCEPTS                            | 5-1   |
| FILE PROCESSING EXAMPLES                            | 5-29  |
| FILE PROCESSING STATEMENTS                          | 5-16  |
| Intercom Input File Processing                      | 7-3   |
| Intercom Output File Processing                     | 7-5   |
| FILE-CODES                                          |       |
| Reserved File-Codes                                 | 9-14  |
| Reserved File-Codes                                 | 9-28  |
| FILE-LIMITS                                         |       |
| FILE-LIMIT(S) Phrase                                | 3-7   |
| FILE-NAME                                           |       |
| File-Name Phrase                                    | 3-2   |
| Level Indicator and File-Name                       | 3-13  |
| FILES                                               |       |
| ASSIGNMENT OF FILES                                 | 5-10  |
| collation files                                     | 9-15  |
| PROCESSING NONLABELED MULTIPLE REEL FILES           | 5-4   |
| PROCESSING OPTIONAL FILES                           | 5-3   |

|                                     |  |       |
|-------------------------------------|--|-------|
| FILLER                              |  |       |
| FILLER                              |  | 4-15  |
| FILLER                              |  | 16-13 |
| FILLER                              |  | 16-4  |
| FINAL                               |  |       |
| FINAL                               |  | 8-7   |
| FIXED OVERLAYABLE                   |  |       |
| Fixed overlayable segments          |  | 15-2  |
| FIXED PERMANENT                     |  |       |
| Fixed permanent segments            |  | 15-2  |
| FIXED-LENGTH                        |  |       |
| Fixed-Length Records                |  | 2-13  |
| FLAGGING                            |  |       |
| NONSTANDARD FEATURE FLAGGING        |  | C-1   |
| FLAGNA                              |  |       |
| FLAGNA option                       |  | C-1   |
| FLOAT DOLLAR SIGN                   |  |       |
| FLOAT DOLLAR SIGN                   |  | 17-6  |
| FOOTING                             |  |       |
| FOOTING phrase                      |  | 8-9   |
| FOR CARDS                           |  |       |
| FOR CARDS Phrase                    |  | 3-5   |
| FOR LISTING                         |  |       |
| FOR LISTING Phrase                  |  | 3-6   |
| FORMAT                              |  |       |
| LOGICAL RECORD FORMAT               |  | 2-12  |
| Transaction Message Format          |  | 7-2   |
| VLR FORMAT Phrase                   |  | 3-10  |
| COMMON OPTIONS IN STATEMENT FORMATS |  | 11-4  |
| COMPUTATIONAL Item Formats          |  | 2-25  |
| DISPLAY Item Formats                |  | 2-24  |
| FORMULAS                            |  |       |
| FORMULAS                            |  | 11-2  |
| FROM                                |  |       |
| FROM mnemonic-name phrase           |  | 6-1   |
| FROM PHRASE                         |  | 5-23  |
| FROM phrase                         |  | 9-10  |
| GENERATE                            |  |       |
| GENERATE statement                  |  | 8-3   |
| GET                                 |  |       |
| GET function                        |  | 5-6   |
| GIN                                 |  |       |
| GIN                                 |  | 6-3   |
| GIVING                              |  |       |
| GIVING                              |  | 15-16 |
| GIVING OPTION                       |  | 9-25  |
| GIVING OPTION                       |  | 9-11  |



|                                                           |  |       |
|-----------------------------------------------------------|--|-------|
| GLAPS                                                     |  |       |
| GLAPS                                                     |  | 6-5   |
| GMAP                                                      |  |       |
| RESERVED GMAP LOCATION SYMBOLS                            |  | F-1   |
| GO TO                                                     |  |       |
| GO TO                                                     |  | 9-10  |
| GO TO                                                     |  | 9-26  |
| GO TO                                                     |  | 15-5  |
| GROUP                                                     |  |       |
| GROUP ENTRIES                                             |  | 8-21  |
| group MOVE                                                |  | 2-26  |
| GROUP ITEM                                                |  |       |
| group item                                                |  | 4-13  |
| group item                                                |  | 4-5   |
| GROUP ITEMS                                               |  | 16-5  |
| Group Items and Elementary Items                          |  | 2-1   |
| GROUPING                                                  |  |       |
| grouping                                                  |  | 12-7  |
| GROUPS                                                    |  |       |
| GROUPS OF ELEMENTARY ITEMS                                |  | 4-20  |
| GTIME                                                     |  |       |
| GTIME                                                     |  | 6-5   |
| HIERARCHY                                                 |  |       |
| hierarchy of data                                         |  | 4-3   |
| HIGH-VALUE                                                |  |       |
| HIGH-VALUE                                                |  | 6-4   |
| HIS                                                       |  |       |
| HIS COPY                                                  |  | 14-1  |
| HIS COPY WITH CCOMDK                                      |  | 14-5  |
| HIS COPY WITH COMDK                                       |  | 14-5  |
| HMS                                                       |  |       |
| HMS option                                                |  | 6-5   |
| I-O                                                       |  |       |
| I-O mode                                                  |  | 5-3   |
| I-O OPTION                                                |  | 5-19  |
| IF                                                        |  |       |
| IF                                                        |  | 12-5  |
| INDEPENDENCE                                              |  |       |
| device independence                                       |  | 5-10  |
| INDEPENDENT                                               |  |       |
| Independent segments                                      |  | 15-2  |
| INDEX DATA ITEM                                           |  |       |
| index data item                                           |  | 12-3  |
| index data item                                           |  | 4-18  |
| Comparisons Involving Index-Names and/or Index Data Items |  | 13-10 |
| index data items                                          |  | 13-3  |
| index data items                                          |  | 13-9  |

|                                                           |  |       |
|-----------------------------------------------------------|--|-------|
| INDEX-NAME                                                |  |       |
| index-name                                                |  | 13-1  |
| index-name                                                |  | 4-18  |
| Comparisons Involving Index-Names and/or Index Data Items |  | 13-10 |
| INDEXED                                                   |  |       |
| INDEXED phrase                                            |  | 4-4   |
| INDEXED BY                                                |  |       |
| INDEXED BY                                                |  | 2-10  |
| INDEXED BY                                                |  | 13-3  |
| INDEXED BY                                                |  | 13-7  |
| INDEXED BY                                                |  | 13-9  |
| INDEXING                                                  |  |       |
| INDEXING                                                  |  | 13-3  |
| INDEXING                                                  |  | 2-10  |
| Rules for Subscripting and Indexing                       |  | 13-4  |
| INDICATOR                                                 |  |       |
| Level Indicator and File-Name                             |  | 3-13  |
| INITIATE                                                  |  |       |
| INITIATE statement                                        |  | 8-3   |
| INPUT                                                     |  |       |
| COBOL INPUT STATEMENT PROCESSING                          |  | 7-5   |
| INPUT mode                                                |  | 5-3   |
| INPUT OPTION                                              |  | 5-19  |
| Intercom Input File Processing                            |  | 7-3   |
| Merge Input Processing                                    |  | 9-25  |
| Sort Input Processing                                     |  | 9-9   |
| System Input                                              |  | 6-3   |
| INPUT PROCEDURE                                           |  |       |
| INPUT PROCEDURE OPTION                                    |  | 9-10  |
| INPUT-OUTPUT                                              |  |       |
| Input-Output Techniques                                   |  | 16-9  |
| INTERFACE                                                 |  |       |
| TPE/TPAP Interface                                        |  | 7-3   |
| INTO                                                      |  |       |
| INTO OPTION                                               |  | 5-21  |
| INTO phrase                                               |  | 9-12  |
| INTO phrase                                               |  | 9-26  |
| INVALID KEY                                               |  |       |
| INVALID KEY PHRASE                                        |  | 5-22  |
| INVALID KEY PHRASE                                        |  | 5-25  |
| ITEMS                                                     |  |       |
| COMPUTATIONAL items                                       |  | 4-15  |
| subordinate items                                         |  | 2-1   |
| JUSTIFIED                                                 |  |       |
| JUSTIFIED                                                 |  | 4-19  |
| JUSTIFIED Clause                                          |  | 4-3   |

|                                     |       |
|-------------------------------------|-------|
| KEY                                 |       |
| KEY phrase                          | 4-5   |
| KEY phrase                          | 13-7  |
| Merge Key Declarations              | 9-22  |
| Merge Key Evaluation                | 9-23  |
| Sort Key Declarations               | 9-5   |
| Sort Key Evaluation                 | 9-7   |
| keys                                | 9-1   |
| LABEL                               |       |
| label check                         | 5-27  |
| LABEL PROCEDURE                     |       |
| LABEL PROCEDURE PHRASE              | 5-27  |
| LABEL RECORDS                       |       |
| LABEL RECORD(S) Clause              | 3-17  |
| LABELED COMMON                      |       |
| Labeled Common                      | 15-13 |
| Labeled Common areas                | 9-19  |
| Labeled Common areas                | 9-29  |
| LANGUAGE                            |       |
| OBSOLETE LANGUAGE ELEMENTS          | 17-1  |
| LEAVING                             |       |
| LEAVING phrase                      | 17-6  |
| LEVEL                               |       |
| Level Indicator and File-Name       | 3-13  |
| level 77 entries                    | 17-3  |
| LEVEL-NUMBER                        |       |
| Level-number 66                     | 4-4   |
| Level-number 77                     | 4-4   |
| Level-number 88                     | 4-4   |
| level-number 66                     | 4-13  |
| level-number 66                     | 2-4   |
| level-number 77                     | 2-3   |
| level-number 88                     | 2-4   |
| level-number 88                     | 4-1   |
| LEVEL-NUMBER/DATA-NAME              |       |
| Level-Number/Data-Name Entries      | 4-3   |
| LEVEL-NUMBERS                       |       |
| Level-Numbers                       | 2-2   |
| LIBCPY                              |       |
| LIBCPY                              | 14-10 |
| LIBRARY                             |       |
| DESCRIPTION OF THE LIBRARY FACILITY | 14-1  |
| library text                        | 14-3  |
| LIMITATIONS                         |       |
| RESET Stack Limitations             | 8-17  |
| SUM Operand Limitations             | 8-16  |
| LINE                                |       |
| Line Counter                        | 8-9   |
| line spacing                        | 5-24  |

|                                     |  |       |
|-------------------------------------|--|-------|
| LINE NUMBER                         |  |       |
| LINE NUMBER                         |  | 8-10  |
| LINE SWITCHING                      |  |       |
| line switching                      |  | 7-8   |
| LINKING                             |  |       |
| Segmentation, Linking, and Loading  |  | 15-6  |
| LIST/DUMP                           |  |       |
| test monitor list/dump              |  | 16-9  |
| LOADING                             |  |       |
| Segmentation, Linking, and Loading  |  | 15-6  |
| LOCATION SYMBOLS                    |  |       |
| RESERVED GMAP LOCATION SYMBOLS      |  | F-1   |
| LOGICAL                             |  |       |
| LOGICAL RECORD FORMAT               |  | 2-12  |
| logical record                      |  | 2-1   |
| LOW-VOLUME                          |  |       |
| LOW-VOLUME DATA TRANSMISSION        |  | 6-1   |
| MACHINE                             |  |       |
| MACHINE WORD                        |  | 2-16  |
| MERGE                               |  |       |
| MERGE                               |  | 9-2   |
| MERGE                               |  | 15-5  |
| MERGE EQUAL KEY RECORD PROCESSING   |  | 9-25  |
| Merge Configuration                 |  | 9-28  |
| Merge Equal Key Procedures          |  | 9-23  |
| Merge Examples                      |  | 9-29  |
| Merge Input Processing              |  | 9-25  |
| Merge Key Declarations              |  | 9-22  |
| Merge Key Evaluation                |  | 9-23  |
| Merge Output Processing             |  | 9-25  |
| The Merge File                      |  | 9-22  |
| MERGING                             |  |       |
| Merging                             |  | 9-1   |
| merging process                     |  | 9-27  |
| MNEMONIC-NAME                       |  |       |
| FROM mnemonic-name phrase           |  | 6-1   |
| UPON mnemonic-name phrase           |  | 6-2   |
| MODE                                |  |       |
| Direct-Access (DAC) Mode Processing |  | 7-8   |
| I-O mode                            |  | 5-3   |
| INPUT mode                          |  | 5-3   |
| OUTPUT mode                         |  | 5-3   |
| MODULARIZATION                      |  |       |
| DESCRIPTION OF MODULARIZATION       |  | 15-11 |
| MODULE                              |  |       |
| Multiple Module Program Example     |  | 15-17 |
| Modules                             |  | 15-12 |

|                                           |       |
|-------------------------------------------|-------|
| MOVE                                      |       |
| EXAMPLES OF MOVE STATEMENTS               | 10-3  |
| group MOVE                                | 2-26  |
| MOVE                                      | 9-10  |
| MOVE SPACES                               | 16-12 |
| MOVE STATEMENT                            | 10-1  |
| MOVE CORRESPONDING                        |       |
| EXAMPLES OF MOVE CORRESPONDING STATEMENTS | 10-7  |
| MOVE CORRESPONDING STATEMENT              | 10-5  |
| MOVE...CORRESPONDING                      |       |
| MOVE...CORRESPONDING                      | 16-14 |
| MULTIPLE FILE                             |       |
| MULTIPLE FILE Phrase                      | 3-11  |
| Multiple file tape positioning            | 5-12  |
| MULTIPLE REEL                             |       |
| PROCESSING NONLABELED MULTIPLE REEL FILES | 5-4   |
| MULTIPLE REEL/UNIT                        |       |
| FOR MULTIPLE REEL/UNIT Phrase             | 3-6   |
| MULTIPLIER                                |       |
| span multiplier                           | 2-28  |
| MULTIPLY                                  |       |
| MULTIPLY Statement                        | 11-9  |
| NEGATIVE                                  |       |
| NEGATIVE                                  | 12-3  |
| NEXT GROUP                                |       |
| NEXT GROUP                                | 8-10  |
| NLSTIN                                    |       |
| NLSTIN option                             | 16-2  |
| NO DATA                                   |       |
| NO DATA phrase                            | 7-5   |
| NO REWIND                                 |       |
| NO REWIND OPTION                          | 5-20  |
| NONSTANDARD                               |       |
| NONSTANDARD FEATURE FLAGGING              | C-1   |
| NOT                                       |       |
| NOT                                       | 12-5  |
| NOT Operator                              | 12-11 |
| NOXREF                                    |       |
| NOXREF option                             | 16-15 |
| NUMBERS                                   |       |
| BINARY NUMBERS                            | 2-18  |
| NUMERIC                                   |       |
| NUMERIC                                   | 17-4  |
| NUMERIC                                   | 12-3  |
| Numeric                                   | 10-2  |
| Numeric Operands                          | 12-2  |

|                                     |  |       |
|-------------------------------------|--|-------|
| NUMERIC EDITED                      |  |       |
| Numeric Edited                      |  | 10-2  |
| OBJECT PROGRAM                      |  |       |
| object program                      |  | 1-3   |
| OCCURRENCE NUMBER                   |  |       |
| occurrence number                   |  | 13-8  |
| occurrence numbers                  |  | 13-2  |
| OCCURS                              |  |       |
| OCCURS                              |  | 13-1  |
| OCCURS                              |  | 2-6   |
| OCCURS                              |  | 9-6   |
| OCCURS                              |  | 9-23  |
| OCCURS Clause                       |  | 4-4   |
| OMITTED                             |  |       |
| OMITTED OPTION                      |  | 3-17  |
| OPEN                                |  |       |
| OPEN Statement                      |  | 5-18  |
| Open Status                         |  | 5-2   |
| OPERAND                             |  |       |
| SUM Operand Limitations             |  | 8-16  |
| Nonnumeric Operands                 |  | 12-2  |
| Numeric Operands                    |  | 12-2  |
| Overlapping Operands                |  | 11-11 |
| OPERATOR                            |  |       |
| NOT Operator                        |  | 12-11 |
| arithmetic operators                |  | 11-2  |
| Unary Operators                     |  | 11-3  |
| OPTIONAL                            |  |       |
| OPTIONAL Phrase                     |  | 3-1   |
| PROCESSING OPTIONAL FILES           |  | 5-3   |
| OPTIONS                             |  |       |
| COMMON OPTIONS IN STATEMENT FORMATS |  | 11-4  |
| SORT-MERGE ELECTIVE OPTIONS         |  | 9-31  |
| system options                      |  | B-1   |
| OR                                  |  |       |
| OR                                  |  | 12-5  |
| ORDER                               |  |       |
| ORDER OF COBOL SOURCE PROGRAM       |  | A-1   |
| Ordering                            |  | 9-1   |
| OUTPUT                              |  |       |
| COBOL OUTPUT STATEMENT PROCESSING   |  | 7-7   |
| Intercom Output File Processing     |  | 7-5   |
| Merge Output Processing             |  | 9-25  |
| OUTPUT mode                         |  | 5-3   |
| Sort Output Processing              |  | 9-11  |
| System Output                       |  | 6-3   |
| OUTPUT PROCEDURE                    |  |       |
| OUTPUT PROCEDURE OPTION             |  | 9-11  |
| OUTPUT PROCEDURE OPTION             |  | 9-26  |
| OVERFLOW                            |  |       |
| Overflow Breaks                     |  | 8-8   |
| overflow condition                  |  | 8-9   |

|                                                |  |       |
|------------------------------------------------|--|-------|
| OVERLAPPING                                    |  |       |
| Overlapping Operands                           |  | 11-11 |
| OVERLAY                                        |  |       |
| Linked Overlay Environment Constraints         |  | 15-16 |
| OVERLAY Phrase                                 |  | 3-1   |
| overlay requirements                           |  | 15-1  |
| PACKED DECIMAL                                 |  |       |
| PACKED DECIMAL EFFICIENCY TECHNIQUES           |  | 16-15 |
| packed decimal                                 |  | 2-32  |
| packed decimal                                 |  | 4-18  |
| PACKED SYNCHRONIZED                            |  |       |
| COMPUTATIONAL-3 PACKED SYNCHRONIZED Data Items |  | 2-32  |
| PADDING                                        |  |       |
| padding                                        |  | 3-5   |
| PAGE                                           |  |       |
| Page Breaks                                    |  | 8-8   |
| Page Counter                                   |  | 8-10  |
| page spacing                                   |  | 5-24  |
| PARENTHESES                                    |  |       |
| parentheses                                    |  | 12-7  |
| parentheses                                    |  | 11-2  |
| PARTITIONED                                    |  |       |
| Partitioned Records                            |  | 2-15  |
| PERFORM                                        |  |       |
| PERFORM                                        |  | 9-26  |
| PERFORM                                        |  | 9-10  |
| PERFORM                                        |  | 15-5  |
| VARYING PHRASE (PERFORM STATEMENT)             |  | 11-14 |
| PERIPHERAL                                     |  |       |
| Peripheral Devices                             |  | 5-12  |
| PHYSICAL                                       |  |       |
| physical record                                |  | 2-1   |
| PICTURE                                        |  |       |
| PICTURE                                        |  | 17-9  |
| PICTURE                                        |  | 17-6  |
| PICTURE                                        |  | 10-1  |
| PICTURE Clause                                 |  | 4-8   |
| PLACES                                         |  |       |
| PLACES option                                  |  | 17-7  |
| POINT LOCATION                                 |  |       |
| POINT LOCATION Clause                          |  | 17-7  |
| POSITION                                       |  |       |
| POSITION option                                |  | 3-11  |
| Multiple file tape positioning                 |  | 5-12  |
| positioning of data                            |  | 4-3   |

|                                                     |  |       |
|-----------------------------------------------------|--|-------|
| POSITIVE                                            |  |       |
| POSITIVE                                            |  | 12-3  |
| PREFERRED                                           |  |       |
| preferred spans                                     |  | 2-28  |
| PREPARED                                            |  |       |
| PREPARED OPTION                                     |  | 17-1  |
| PRIORITY-NUMBERS                                    |  |       |
| PRIORITY-NUMBERS                                    |  | 15-3  |
| PROCEDURE DIVISION                                  |  |       |
| PROCEDURE DIVISION SYMBOLS                          |  | F-2   |
| Procedure Division Communications                   |  | 15-13 |
| PROCEDURES                                          |  |       |
| CONDITIONAL PROCEDURES                              |  | 12-1  |
| Merge Equal Key Procedures                          |  | 9-23  |
| Sort Equal Key Procedures                           |  | 9-8   |
| PROCESS AREA                                        |  |       |
| APPLY PROCESS AREA phrase                           |  | 16-9  |
| PROCESS AREA Phrase                                 |  | 3-8   |
| process area                                        |  | 4-6   |
| process area                                        |  | 5-14  |
| PROCESSING                                          |  |       |
| COBOL INPUT STATEMENT PROCESSING                    |  | 7-5   |
| COBOL OUTPUT STATEMENT PROCESSING                   |  | 7-7   |
| Direct-Access (DAC) Mode Processing                 |  | 7-8   |
| MERGE EQUAL KEY RECORD PROCESSING                   |  | 9-25  |
| Merge Input Processing                              |  | 9-25  |
| Merge Output Processing                             |  | 9-25  |
| PROCESSING NONLABELED MULTIPLE REEL FILES           |  | 5-4   |
| PROCESSING OPTIONAL FILES                           |  | 5-3   |
| Random-Access Processing                            |  | 5-2   |
| Sequential-Access Processing                        |  | 5-1   |
| SORT EQUAL KEY RECORD PROCESSING                    |  | 9-9   |
| Sort Input Processing                               |  | 9-9   |
| Sort Output Processing                              |  | 9-11  |
| PROCESSING MODE                                     |  |       |
| PROCESSING MODE Phrase                              |  | 3-8   |
| PROCESSOR                                           |  |       |
| Processor Time                                      |  | 6-5   |
| Processor Time                                      |  | 16-6  |
| PROGRAM                                             |  |       |
| DATA TRANSMISSION PROGRAM EXAMPLE                   |  | 6-9   |
| Multiple Module Program Example                     |  | 15-17 |
| REPORT WRITER PROGRAM EXAMPLE                       |  | 8-27  |
| Structure of Program Segments                       |  | 15-3  |
| Transaction Processing Applications Program Example |  | 7-9   |
| PROGRAM-ID                                          |  |       |
| PROGRAM-ID                                          |  | 15-13 |
| QUALIFICATION                                       |  |       |
| Qualification                                       |  | 2-4   |
| RANDOM-ACCESS                                       |  |       |
| Random-Access Processing                            |  | 5-2   |



|                     |                                          |       |
|---------------------|------------------------------------------|-------|
| RANGE               |                                          |       |
|                     | RANGE Clause                             | 17-7  |
| RD                  |                                          |       |
|                     | RD Entries                               | 8-5   |
| READ                |                                          |       |
|                     | READ Statement                           | 5-20  |
| RECORD              |                                          |       |
|                     | LOGICAL RECORD FORMAT                    | 2-12  |
|                     | logical record                           | 2-1   |
|                     | MERGE EQUAL KEY RECORD PROCESSING        | 9-25  |
|                     | physical record                          | 2-1   |
|                     | Record Areas                             | 5-14  |
|                     | SAME RECORD AREA                         | 5-15  |
|                     | SAME RECORD AREA Phrase                  | 3-10  |
|                     | SAME RECORD AREA phrase                  | 16-10 |
|                     | SORT EQUAL KEY RECORD PROCESSING         | 9-9   |
| RECORD CONTAINS     |                                          |       |
|                     | RECORD CONTAINS Clause                   | 3-21  |
| RECORD DESCRIPTIONS |                                          |       |
|                     | RECORD DESCRIPTIONS                      | 4-1   |
| RECORDING MODE      |                                          |       |
|                     | RECORDING MODE Clause                    | 3-21  |
| RECORDS             |                                          |       |
|                     | Constant Records                         | 17-3  |
|                     | Fixed-Length Records                     | 2-13  |
|                     | Partitioned Records                      | 2-15  |
|                     | Variable-Length Records                  | 2-13  |
| RECOVERY            |                                          |       |
|                     | spill recovery                           | 9-16  |
| REDEFINES           |                                          |       |
|                     | REDEFINES                                | 16-5  |
|                     | REDEFINES Clause                         | 4-10  |
|                     | REDEFINES Entries                        | 2-3   |
| REEL                |                                          |       |
|                     | STANDARD CLOSE REEL                      | 5-26  |
| RELATION            |                                          |       |
|                     | Abbreviated Combined Relation Conditions | 12-8  |
|                     | RELATION CONDITION                       | 12-1  |
| RELEASE             |                                          |       |
|                     | RELEASE Statement                        | 9-10  |
| REMOTE              |                                          |       |
|                     | REMOTE                                   | 6-4   |
|                     | Remote Devices                           | 6-4   |
| RENAMES             |                                          |       |
|                     | RENAMES Clause                           | 4-13  |
|                     | RENAMES Entries                          | 2-4   |
| RENAMING            |                                          |       |
|                     | RENAMING                                 | 3-12  |
|                     | RENAMING Phrase                          | 3-3   |
|                     | RENAMING phrase                          | 14-5  |

|                                        |  |       |
|----------------------------------------|--|-------|
| REPLACING                              |  |       |
| REPLACING PHRASE (EXAMINE STATEMENT)   |  | 10-8  |
| REPORT                                 |  |       |
| Report Control                         |  | 8-2   |
| Report Table Capacity                  |  | 8-19  |
| summary report                         |  | 16-6  |
| REPORT GROUP                           |  |       |
| CALCULATION OF REPORT GROUP SIZE       |  | 8-25  |
| REPORT GROUP ENTRIES                   |  | 8-20  |
| Report Group Entries                   |  | 8-5   |
| Report Groups                          |  | 8-6   |
| REPORT PRINTING                        |  |       |
| Incremental Report Printing Techniques |  | 16-10 |
| REPORT WRITER                          |  |       |
| DESCRIPTION OF THE REPORT WRITER       |  | 8-1   |
| REPORT WRITER EFFICIENCY TECHNIQUES    |  | 8-10  |
| REPORT WRITER PROGRAM EXAMPLE          |  | 8-27  |
| REPORT WRITER SYMBOLS                  |  | F-2   |
| REPORT WRITER TABLE CONSTRAINTS        |  | 8-16  |
| REPORTS                                |  |       |
| REPORT(S) Clause                       |  | 3-23  |
| RERUN                                  |  |       |
| RERUN Phrase                           |  | 3-10  |
| RESERVE                                |  |       |
| RESERVE Phrase                         |  | 3-6   |
| RESERVE phrase                         |  | 5-14  |
| RESERVED GMAP LOCATION SYMBOLS         |  | F-1   |
| Reserved File-Codes                    |  | 9-14  |
| Reserved File-Codes                    |  | 9-28  |
| RESET                                  |  |       |
| RESET Stack Limitations                |  | 8-17  |
| RESOURCE                               |  |       |
| Resource Allocation                    |  | 16-5  |
| RESULTANT-IDENTIFIER                   |  |       |
| resultant-identifier                   |  | 11-4  |
| RETURN                                 |  |       |
| RETURN Statement                       |  | 9-26  |
| RETURN Statement                       |  | 9-12  |
| ROLLING FORWARD                        |  |       |
| ROLLING FORWARD                        |  | 8-11  |
| ROUNDED                                |  |       |
| ROUNDED Option                         |  | 11-4  |
| SAME                                   |  |       |
| SAME AREA                              |  | 5-14  |
| SAME AREA Phrase                       |  | 3-10  |
| SAME AREA phrase                       |  | 16-9  |
| SAME RECORD AREA                       |  | 5-15  |
| SAME RECORD AREA Phrase                |  | 3-10  |
| SAME RECORD AREA phrase                |  | 16-10 |
| SAME SORT or SORT-MERGE AREA Phrase    |  | 3-11  |

|                                     |  |       |
|-------------------------------------|--|-------|
| SD                                  |  |       |
| SD                                  |  | 9-22  |
| SD                                  |  | 9-2   |
| SEARCH                              |  |       |
| SEARCH                              |  | 4-18  |
| SEARCH and SET Statement Examples   |  | 13-10 |
| SEARCH STATEMENT                    |  | 13-7  |
| SECTION                             |  |       |
| CONSTANT SECTION                    |  | 17-2  |
| Sections                            |  | 15-12 |
| SEEK                                |  |       |
| SEEK Statement                      |  | 5-25  |
| SEGMENT                             |  |       |
| Segment Classification              |  | 15-2  |
| SEGMENT-LIMIT                       |  |       |
| SEGMENT-LIMIT                       |  | 15-4  |
| SEGMENTATION                        |  |       |
| DESCRIPTION OF SEGMENTATION         |  | 15-1  |
| Effects of Segmentation on Listings |  | 15-10 |
| Segmentation Control                |  | 15-3  |
| Segmentation, Linking, and Loading  |  | 15-6  |
| SEGMENTS                            |  |       |
| Fixed overlayable segments          |  | 15-2  |
| Fixed permanent segments            |  | 15-2  |
| Independent segments                |  | 15-2  |
| Structure of Program Segments       |  | 15-3  |
| SEGMNT                              |  |       |
| SEGMNT option                       |  | 15-3  |
| SELECT                              |  |       |
| SELECT SENTENCE                     |  | 3-1   |
| SELECT sentence                     |  | 9-2   |
| SELECT sentence                     |  | 9-22  |
| SELECT sentence                     |  | 5-1   |
| SEQUENCED                           |  |       |
| SEQUENCED Clause                    |  | 17-8  |
| SEQUENTIAL-ACCESS                   |  |       |
| Sequential-Access Processing        |  | 5-1   |
| SET                                 |  |       |
| SEARCH and SET Statement Examples   |  | 13-10 |
| SET                                 |  | 4-18  |
| SET                                 |  | 2-10  |
| SET Statement Rules                 |  | 13-9  |
| SIGN                                |  |       |
| SIGN CONDITION                      |  | 12-3  |
| SIGNED Clause                       |  | 17-8  |
| SIMPLE                              |  |       |
| Simple Conditions                   |  | 12-1  |
| SIZE                                |  |       |
| CALCULATION OF REPORT GROUP SIZE    |  | 8-25  |
| SIZE Clause                         |  | 17-9  |

|                                         |  |       |
|-----------------------------------------|--|-------|
| SIZE ERROR                              |  |       |
| SIZE ERROR Option                       |  | 11-5  |
| SLEW                                    |  |       |
| slew control                            |  | 6-4   |
| slew control                            |  | 5-23  |
| SORT                                    |  |       |
| SAME SORT or SORT-MERGE AREA Phrase     |  | 3-11  |
| SORT                                    |  | 9-2   |
| SORT                                    |  | 15-5  |
| SORT EQUAL KEY RECORD PROCESSING        |  | 9-9   |
| Sort Areas and Sort-Merge Areas         |  | 5-16  |
| Sort Configuration                      |  | 9-17  |
| Sort Equal Key Procedures               |  | 9-8   |
| Sort Examples                           |  | 9-20  |
| Sort Input Processing                   |  | 9-9   |
| Sort Key Declarations                   |  | 9-5   |
| Sort Key Evaluation                     |  | 9-7   |
| Sort Output Processing                  |  | 9-11  |
| The Sort File                           |  | 9-4   |
| SORT-MERGE                              |  |       |
| SAME SORT or SORT-MERGE AREA Phrase     |  | 3-11  |
| SORT-MERGE ELECTIVE OPTIONS             |  | 9-31  |
| SORT-MERGE FILE DESCRIPTION ENTRIES     |  | 3-13  |
| Sort Areas and Sort-Merge Areas         |  | 5-16  |
| SORTING                                 |  |       |
| Sorting                                 |  | 9-1   |
| sorting process                         |  | 9-14  |
| SOURCE                                  |  |       |
| SOURCE clause                           |  | 8-10  |
| SOURCE ENTRIES                          |  | 8-21  |
| SOURCE PROGRAM                          |  |       |
| ORDER OF COBOL SOURCE PROGRAM           |  | A-1   |
| source program                          |  | 1-3   |
| SOURCE/SUM                              |  |       |
| SOURCE/SUM Correlation                  |  | 8-13  |
| SPACE-SAVING                            |  |       |
| TIME-SAVING AND SPACE-SAVING TECHNIQUES |  | 16-9  |
| SPACES                                  |  |       |
| MOVE SPACES                             |  | 16-12 |
| SPACES                                  |  | 16-5  |
| SPACING                                 |  |       |
| line spacing                            |  | 5-24  |
| page spacing                            |  | 5-24  |
| SPAN                                    |  |       |
| span conversion                         |  | 2-28  |
| span multiplier                         |  | 2-28  |
| preferred spans                         |  | 2-28  |
| SPECIAL-NAMES                           |  |       |
| SPECIAL-NAMES                           |  | 6-1   |
| SPILL                                   |  |       |
| spill recovery                          |  | 9-16  |

|                                           |  |       |
|-------------------------------------------|--|-------|
| SPINOFF                                   |  |       |
| spinoff feature                           |  | 16-10 |
| STANDARD                                  |  |       |
| STANDARD CLOSE FILE                       |  | 5-26  |
| STANDARD CLOSE REEL                       |  | 5-26  |
| STANDARD COLLATING SEQUENCE               |  | D-1   |
| STANDARD OPTION                           |  | 3-17  |
| STATEMENTS                                |  |       |
| ACCEPT STATEMENTS                         |  | 6-1   |
| ARITHMETIC STATEMENTS                     |  | 11-6  |
| CONDITIONAL STATEMENTS                    |  | 17-10 |
| DEBUG STATEMENTS                          |  | 16-1  |
| DISPLAY STATEMENTS                        |  | 6-2   |
| EXAMPLES OF MOVE CORRESPONDING STATEMENTS |  | 10-7  |
| EXAMPLES OF MOVE STATEMENTS               |  | 10-3  |
| FILE PROCESSING STATEMENTS                |  | 5-16  |
| Multiple Results in Arithmetic Statements |  | 11-10 |
| STATUS                                    |  |       |
| Closed Status                             |  | 5-2   |
| Open Status                               |  | 5-2   |
| STORAGE                                   |  |       |
| Noncontiguous Constant Storage            |  | 17-2  |
| STRUCTURE                                 |  |       |
| Structure of Program Segments             |  | 15-3  |
| DATA STRUCTURES                           |  | 2-1   |
| SUBORDINATE                               |  |       |
| subordinate items                         |  | 2-1   |
| SUBSCRIPTING                              |  |       |
| Rules for Subscribing and Indexing        |  | 13-4  |
| SUBSCRIPTING                              |  | 2-9   |
| SUBSCRIPTING                              |  | 13-2  |
| SUBTOTALLING                              |  |       |
| SUBTOTALLING                              |  | 8-11  |
| SUBTRACT                                  |  |       |
| SUBTRACT Statement                        |  | 11-10 |
| SUM                                       |  |       |
| SUM Counter Manipulation                  |  | 8-10  |
| SUM ENTRIES                               |  | 8-23  |
| SUM Operand Limitations                   |  | 8-16  |
| SUMMARY                                   |  |       |
| summary report                            |  | 16-6  |
| SWITCH-STATUS                             |  |       |
| SWITCH-STATUS CONDITION                   |  | 12-5  |
| SWITCHES                                  |  |       |
| Switches                                  |  | 6-7   |
| SYMBOL PAIRS                              |  |       |
| Symbol Pairs                              |  | 11-3  |

|                                     |       |
|-------------------------------------|-------|
| SYMBOLS                             |       |
| NONFILE DATA SYMBOLS                | F-1   |
| PROCEDURE DIVISION SYMBOLS          | F-2   |
| REPORT WRITER SYMBOLS               | F-2   |
| SYMBOLS USING 'FC'                  | F-1   |
| UTILITY SYMBOLS                     | F-3   |
| SYMDEFS                             |       |
| SYMDEFS                             | 5-5   |
| SYNCHRONIZED                        |       |
| SYNCHRONIZED                        | 16-13 |
| SYNCHRONIZED LEFT                   |       |
| SYNCHRONIZED LEFT                   | 4-14  |
| SYNCHRONIZED RIGHT                  |       |
| SYNCHRONIZED RIGHT                  | 4-14  |
| SYSOUT                              |       |
| SYSOUT                              | 6-3   |
| SYSOUT                              | 16-6  |
| SYSTEM                              |       |
| System Input                        | 6-3   |
| System Output                       | 6-3   |
| system options                      | B-1   |
| SYSTEM STANDARD FORMAT              |       |
| SYSTEM STANDARD FORMAT Phrase       | 3-9   |
| System Standard Format              | 5-10  |
| TABLE                               |       |
| REPORT WRITER TABLE CONSTRAINTS     | 8-16  |
| Report Table Capacity               | 8-19  |
| TPAP Profile Table                  | 7-1   |
| TABLE ELEMENT                       |       |
| table element                       | 13-7  |
| table element                       | 13-1  |
| TABLE HANDLING                      |       |
| DESCRIPTION OF TABLE HANDLING       | 13-1  |
| TABLE RESIDUE                       |       |
| table residue                       | 4-6   |
| TABLES                              |       |
| Tables of Constants                 | 17-3  |
| Tables of Data Items                | 2-6   |
| TALLYING                            |       |
| TALLYING PHRASE (EXAMINE STATEMENT) | 11-12 |
| TAPE                                |       |
| Multiple file tape positioning      | 5-12  |

|                                                      |       |
|------------------------------------------------------|-------|
| TECHNIQUES                                           |       |
| COMPILATION TECHNIQUES                               | 16-2  |
| DATA TRANSMISSION TECHNIQUES                         | 6-3   |
| Data Description Techniques                          | 16-13 |
| Data Manipulation Techniques                         | 16-12 |
| EFFICIENCY TECHNIQUES                                | 16-1  |
| Incremental Report Printing Techniques               | 16-10 |
| Input-Output Techniques                              | 16-9  |
| PACKED DECIMAL EFFICIENCY TECHNIQUES                 | 16-15 |
| REPORT WRITER EFFICIENCY TECHNIQUES                  | 8-10  |
| TIME-SAVING AND SPACE-SAVING TECHNIQUES              | 16-9  |
| TERMINATE                                            |       |
| TERMINATE statement                                  | 8-3   |
| TEST MONITOR                                         |       |
| test monitor list/dump                               | 16-9  |
| TEXT                                                 |       |
| duplicate text                                       | 4-2   |
| library text                                         | 14-3  |
| TIME                                                 |       |
| Date and Time                                        | 6-5   |
| Processor Time                                       | 6-5   |
| Processor Time                                       | 16-6  |
| TIME-SAVING                                          |       |
| TIME-SAVING AND SPACE-SAVING TECHNIQUES              | 16-9  |
| TOP                                                  |       |
| TOP                                                  | 5-24  |
| TPAP                                                 |       |
| TPAP Profile Table                                   | 7-1   |
| TRANSACTION PROCESSING APPLICATIONS PROGRAMS (TPAPS) | 7-3   |
| TPE                                                  |       |
| TRANSACTION PROCESSING EXECUTIVE (TPE)               | 7-1   |
| TRANSACTION PROCESSING                               |       |
| TRANSACTION PROCESSING APPLICATIONS PROGRAMS (TPAPS) | 7-3   |
| TRANSACTION PROCESSING EXECUTIVE (TPE)               | 7-1   |
| Transaction Processing                               | 6-1   |
| Transaction Processing Abort Codes                   | E-4   |
| Transaction Processing Applications Program Example  | 7-9   |
| TRANSFER                                             |       |
| Transfer of Control                                  | 15-5  |
| TRUTH VALUE                                          |       |
| truth value                                          | 12-1  |
| TYPE                                                 |       |
| TYPE clause                                          | 8-7   |
| TYPEWRITER                                           |       |
| TYPEWRITER                                           | 6-6   |
| UNARY                                                |       |
| Unary Operators                                      | 11-3  |
| UP BY                                                |       |
| UP BY                                                | 13-9  |

|                                    |  |       |
|------------------------------------|--|-------|
| UPON                               |  |       |
| UPON mnemonic-name phrase          |  | 6-2   |
| USAGE                              |  |       |
| USAGE Clause                       |  | 4-17  |
| USAGE INDEX                        |  |       |
| USAGE INDEX                        |  | 2-11  |
| USAGE IS INDEX                     |  |       |
| USAGE IS INDEX                     |  | 4-18  |
| USE                                |  |       |
| USE                                |  | 5-19  |
| USE                                |  | 5-22  |
| USE                                |  | 5-23  |
| USE Statement                      |  | 5-26  |
| USE BEFORE REPORTING               |  |       |
| USE BEFORE REPORTING               |  | 8-17  |
| USE BEFORE REPORTING PHRASE        |  | 5-28  |
| USING                              |  |       |
| SYMBOLS USING 'FC'                 |  | F-1   |
| USING                              |  | 15-16 |
| USING OPTION                       |  | 9-9   |
| UTILITY                            |  |       |
| UTILITY SYMBOLS                    |  | F-3   |
| VALUE                              |  |       |
| VALUE Clause                       |  | 4-19  |
| VALUE clause                       |  | 17-3  |
| VALUE ENTRIES                      |  | 8-24  |
| VALUE OF                           |  |       |
| VALUE OF Clause                    |  | 3-23  |
| VARIABLE-LENGTH                    |  |       |
| Variable-Length Records            |  | 2-13  |
| VARYING                            |  |       |
| VARYING PHRASE (PERFORM STATEMENT) |  | 11-14 |
| VARYING phrase                     |  | 13-7  |
| VERTICAL POSITIONING               |  |       |
| vertical positioning               |  | 5-23  |
| VLR                                |  |       |
| VLR FORMAT Phrase                  |  | 3-10  |
| WORD                               |  |       |
| MACHINE WORD                       |  | 2-16  |
| WRITE                              |  |       |
| WRITE Statement                    |  | 5-22  |
| ZERO                               |  |       |
| ZERO                               |  | 12-3  |
| ZERO SUPPRESS                      |  |       |
| ZERO SUPPRESS                      |  | 17-6  |





**HONEYWELL INFORMATION SYSTEMS**  
Technical Publications Remarks Form

CUT ALONG L

TITLE

SERIES 60(LEVEL 66)/6000 COBOL USER'S GUIDE  
ADDENDUM B

ORDER NO.

DD26B, REV. 0

DATED

JULY 1977

**ERRORS IN PUBLICATION**

[Empty box for reporting errors in publication]

**SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION**

[Empty box for providing suggestions for improvement to publication]



Your comments will be promptly investigated by appropriate technical personnel and action will be taken as required. If you require a written reply, check here and furnish complete mailing address below.

FROM: NAME \_\_\_\_\_

DATE \_\_\_\_\_

TITLE \_\_\_\_\_

COMPANY \_\_\_\_\_

ADDRESS \_\_\_\_\_

\_\_\_\_\_

CUT ALONG LINE

PLEASE FOLD AND TAPE -

NOTE: U.S. Postal Service will not deliver stapled forms

FIRST CLASS  
PERMIT NO. 39531  
WALTHAM, MA  
02154

Business Reply Mail  
Postage Stamp Not Necessary if Mailed in the United States

Postage Will Be Paid By:

HONEYWELL INFORMATION SYSTEMS  
200 SMITH STREET  
WALTHAM, MA 02154

ATTENTION: PUBLICATIONS, MS 486

**Honeywell**